STEPS TOWARD A COMPILER FOR BLISS-360

Richard Charles Bahler

NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

STEPS TOWARD A COMPILER FOR BLISS-360

by

Richard Charles Bahler

Thesis Advisor: Gary A. Kildall

June 1972

Approved for public release; distribution unlimited.

Steps Toward a Compiler for BLISS-360

by

Richard Charles Bahler
Major, United States Marine Corps Reserve
B.A., University of Rochester, 1955

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1972

ABSTRACT

The design of a compiler for the IBM S/360 systems implementation language BLISS-360, a modification of the PDP-10 language BLISS-10, is described. The compiler has a two-pass structure that is based upon the XPL Compiler Generator System. The first of these passes, which uses the XPL prototype compiler Skeleton, is examined in some detail. Fundamental data structures are described for this pass, including a constant table, a dictionary for variable definitions, and an intermediate language table to retain the source program structure and semantics. Modifications which allow the Skeleton compiler to perform a syntax analysis of BLISS-360 programs are discussed and demonstrated.

General requirements are defined for the functions to be performed by the second pass, including machine language code generation from the intermediate language, storage allocation and building program interface linkage.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF DRAWINGS

ACKNOWLEDGEMENT

A word of thanks to Drs. Wulf and Habermann of the Computer Science Department of Carnegie-Mellon University for providing the documentation for BLISS-10.

I. INTRODUCTION

An IBM S/360 version of the Basic Language for the Implementation of System Software, BLISS-360, has been designed by Zavoyski [Ref. 1] from the BLISS language for the PDP-10 [Ref. 2]. The next task then is to design and build a compiler for this language. With this objective in mind, the ultimate goal of the project and the specific objectives of this thesis are stated.

A. PROJECT GOAL

In order for BLISS-360 to become a viable tool for the production of S/360 software programs it is necessary to have for it a compiler which will execute efficiently and produce effective machine language. The production of this compiler is the ultimate project goal.

B. THESIS OBJECTIVES

The primary thesis objective is to provide a basic structure for accomplishing the goal. This was planned as three basic phases.

First, define the approach by examining several alternative methods of compiler construction to select one suitable for the language and convenient to implement.

Second, design the compiler. Based on the approach selected (a two-pass compiler using the XPL Compiler Generator System [Ref. 3]) this phase was divided into two subgoals; first, provide a detailed design of pass one and second, provide the general requirements for pass two.

Third, provide sufficient programming for pass one (a modification of the XPL Skeleton Compiler [Ref. 3]) to allow the syntax analysis of programs written in BLISS-360.

II. A BLISS-360 COMPILER -- POSSIBLE APPROACHES

In the effective construction of a compiler it is necessary first to examine the objectives intended for the compiler and then to balance these objectives against the available construction techniques.

A. REQUIREMENTS OF THE COMPILER

The principal intent of BLISS-360 is to produce production quality systems programs. Its compiler must therefore first meet the requirement of efficient code production.

System programs provide essential services in the management and use of computer resources. In general, however, they are part of the operating system overhead in terms of both time and storage space. Thus, it is essential that a compiler for the production of system programs approach the efficiency of good assembly language programming in its production of machine language.

As an extension of the first requirement, run-time storage must be managed effectively. Mechanisms must be established to keep central memory requirements to a minimum, and relinquish unused areas of memory for future use by other blocks of the program.

A paged environment places the additional burden of maintaining contiguity of data wherever possible in order to minimize page faults.

BLISS-360 programs must be able to communicate with existing operating systems, in particular OS/360 and CP/CMS. They must also

be able to operate without the support of an operating system. Thus, multiple forms of interface may need to be created.

In order to maintain its effectiveness, a compiler must be adaptable to changes in the language it implements and to changing requirements for its object code.

B. COMPILER CONSTRUCTION TECHNIQUES CONSIDERED

Several possible ways of building a BLISS-360 compiler were evaluated against the requirements of Sec. II.A. In addition, they were examined to see whether a significant start could be made on a useful compiler in a limited time. The techniques evaluated were as follows.

1. Rewrite BLISS

One of the existing two versions of BLISS (for the PDP-10 and the PDP-11) could be rewritten, incorporating the changes for the S/360, in an existing S/360 language. This would have the advantage of working with an established BLISS system which has implementation information available [Ref. 2] .

Several languages are available on the S/360 for the rewrite task. Examples of these languages include XPL [Ref. 2] , PL360 [Ref. 4] and S/360 Assembler [Ref. 5] .

The existing BLISS compilers, however, are quite large and complex and would require a significant education period before anything worthwhile could be accomplished. Furthermore, the differences in architecture between the PDP-10 and the S/360 cause many language changes

[Ref. 1] . In addition, these differences would create the need for modified storage management techniques.

This approach was discarded since the size of the effort involved before any return could be seen was too large.

2. Write a New Compiler

The idea of writing a completely new compiler was briefly considered. This approach has some advantages over a complete rewrite in that one is burdened neither with the idiosyncrasies of an existing program nor with the problem of working with two languages at once.

This approach was not taken because of the large amount of time required to get a useful start on a project of this magnitude.

3. Bootstrap - PDP-10 to S/360

Given the availability of a PDP-10, it is possible to bootstrap BLISS from the PDP-10 to the S/360. The BLISS-10 compiler, which is written in BLISS-10, could be modified to produce S/360 machine language (ML) for a subset of its functions sufficient to describe the compiler. Language changes could mostly be avoided at this point as the PDP-10 has a similar structure to the fixed point subset of the S/360 [Ref. 6 and 7] .

Using this BLISS-10 to S/360 ML compiler to compile BLISS-10 on the PDP-10, one could produce a BLISS-10 to S/360 ML compiler which would execute on the S/360. At this point independence from the PDP-10 would be achieved and modifications to BLISS-10 to produce BLISS-360 could be accomplished by further bootstrapping on the S/360. The T-diagram in Fig. 1 illustrates this process.

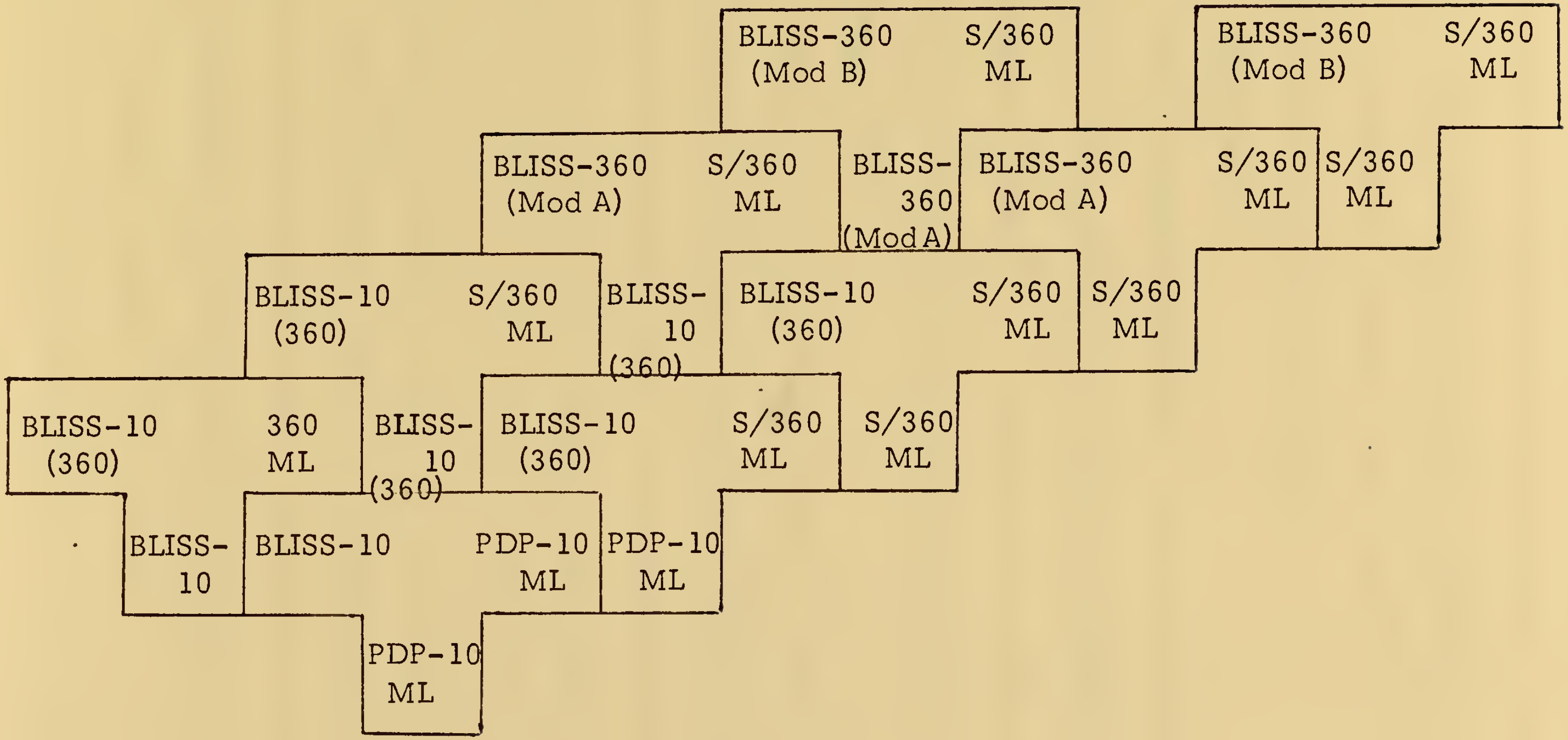


Fig. 1

T-Diagram -- BLISS-10 to BLISS-360

An additional advantage to this approach is that the compiler is maintained in its own language. This reduces the number of languages with which a programmer must be intimately familiar, and improves the modifiability of the compiler since BLISS is well suited to compiler writing.

This would have been the most logical technique; unfortunately, a PDP-10 was not available for this project.

4. XPL Systems

The final method considerd, and the one which was eventually considered to be most suitable, was to make use of the XPL Compiler Generator System [Ref. 3] . For convenience, this system will be referred to as the XPL System since it is written in the PL/1 derivative programming language XPL.

The XPL System operates as follows. The syntax of the source language being compiled is defined in Backus-Naur Form (BNF) so that it is acceptable to McKeeman's Mixed Strategy Precedence parsing technique. This BNF is processed by a BNF analyzer program which generates tables describing the terminal and non-terminal symbols of the language. The tables also define stacking and context checking decisions.

These tables are added to a program called the Skeleton. The Skeleton is a prototype compiler which contains the scanning routines and the basic mechanisms for syntax analysis of a program written in the source language. The syntax analysis routine also calls an empty code synthesis procedure each time a reduction is performed on the source language.

a. One-Pass

In a one-pass XPL system the code synthesis procedure Synthesize is responsible for building the symbol table or dictionary, looking up definitions, generating object code (eg. S/360 ML), and setting up run-time storage.

The one-pass system meets most of the requirements set forth in Sec. II.A. Its greatest asset is its relative ease of initial construction, its modifiability, and its speed of execution.

Once the BNF has been well defined and the basic Synthesize functions written (eg., symbol lookup, and object code generation for the various S/360 instruction formats [Ref. 7]), code synthesis constructions can be added gradually. A basic subset can first be defined in order to demonstrate the basic concepts. The more sophisticated constructions can then be added, or previously defined constructions can be modified.

The one-pass system has many features to recommend it, but it was rejected for reasons which will be noted in Sec. II.C.

b. Multi-Pass

A multi-pass compiler produced using the XPL System uses the same basic mechanism as the single-pass system except that it is split into more than one segment, each of which scans the entire program or a modified form of the program.

(1) Three-Pass. An example of a multi-pass system might be a three-pass system. The first pass would analyze a program

only for block structure and declarations. It would completely build the data dictionary thereby simplifying the remaining grammar by removing potential data conflict problems such as forward function calls and label branches.

The output of the first pass would be the original program without the declarations. This first pass could be built from a Skeleton and an abbreviated grammar, but would most likely consist primarily of an exotic scan routine.

The second pass would analyze the modified program produced by the first pass using essentially the form of the one-pass system, but with these differences: the BNF would not contain declarations, and variable lookup would be performed by the scanner instead of by Synthesize. Synthesize would no longer produce object code but would instead produce an intermediate language (IL) form of the program.

Finally, the third pass would use the IL and the data dictionary to produce machine language (ML) and set up run-time storage and external interface requirements.

(2) More than Three Passes. More passes could be added to further subdivide the functions of the existing passes in order to reduce the central memory requirements of each pass, or to provide some additional capability such as improved code optimization.

(3) Two-Pass. Another variant of the multi-pass system has just two passes. A logical form for this would be to combine the first two passes of the three-pass system mentioned above. Thus, as

shown in Fig. 2, the first pass would scan the source statements, analyze their syntactical structure, perform a semantic analysis, and synthesize the IL. It would also produce dictionary and constant tables for use in the second pass.

The second pass performs the same functions as the third pass of the three-pass system, producing an object module ready to be linked with other modules before loading for execution.

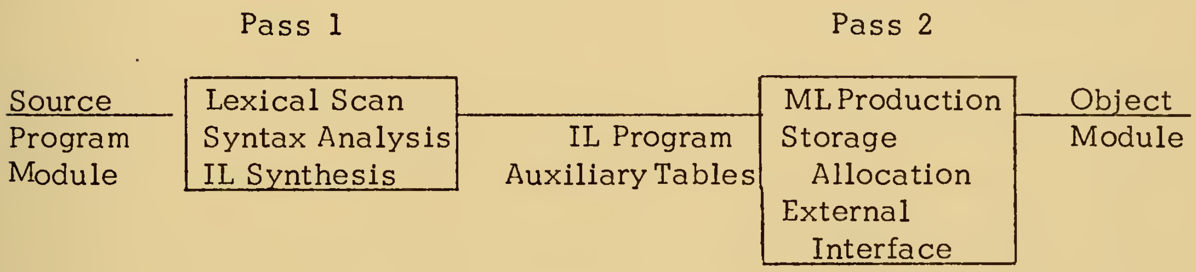


Fig. 2. Two-Pass Compiler

C. THE APPROACH SELECTED

After consideration of the above techniques, a two-pass XPL system was selected as being most appropriate.

The two-pass system will be somewhat slower than an equivalent one-pass system. However, it should more than compensate for this through its potential for code optimization in the second pass, along with reduced central memory requirements for each pass during compilation. Having two passes also improves modularity through isolation of the analysis and synthesis functions from the detailed code generation, storage assignment, and interface handling functions.

It should be noted that the three-pass system described above is not particularly suitable for a BLISS compiler as the declarations can contain or directly refer to expressions, thus making them very difficult to isolate from the rest of the language.

III. DESIGN OF THE TWO-PASS XPL SYSTEM

The general form of a compiler generated using the XPL System, and of a two-pass system in particular, is outlined in Sec. II.B.4. Specific design requirements for the two-pass system will not be defined.

A. PASS 1

The overall structure of Pass 1 is similar to the Skeleton program [Ref. 3] . All of the additions and modifications required to make Pass 1 completely operational are discussed here although, except for the BNF tables, only those detailed in Sec. IV are implemented at present.

1. BNF

The tables generated by processing the BLISS-360 BNF with the XPL BNF analyzer program [Ref. 1] replace the existing tables in the XPL Skeleton compiler.

2. Scanning

The Scan routine of the Skeleton requires extensive reprogramming to recognize several new constant formats, bypass comments in two new formats, recognize macros for storage and retrieval and place character strings and hexadecimal numbers in the constant table.

3. Constants

Pass 1 requires a different technique for handling constants than that presented in the Skeleton for two reasons. First, BLISS-360 recognizes any type of constant that the S/360 can manipulate. This

can be resolved by having Scan recognize the type of constant, then invoking a routine to convert it to internal format.

Second, since the values of constants are not used immediately in a two-pass system, constant tables must be constructed to save their values for the second pass.

4. Identifiers

Skeleton has no built-in identifier processing except for the process of identifying reserved words in Scan. It is therefore necessary to establish a data dictionary and routines for entering and looking-up identifiers.

Dictionary access must be structured so that it reflects the block structured scope of variables of BLISS-360 and at the same time allows pertinent identifier information to be retained for Pass 2.

5. Intermediate Language

A suitable form of intermediate language (IL) must be developed to represent the detailed semantics of the source program and yet be directly translatable into machine language by Pass 2 without further analysis.

A Synthesize routine is then developed to replace the empty Synthesize routine in Skeleton. This new Synthesize routine produces the appropriate IL whenever a parse reduction occurs. As a related function, Synthesize will also look-up and enter identifiers in the dictionary.

6. Miscellaneous

Several minor modifications are mentioned here. Some of these are required and some are merely useful additions which are not essential to the system.

The Initialize routine of the Skeleton must be modified to allow for different terminal symbols, comments and data types.

Output routines will be required to allow the blocking of the various vectors which comprise the constant, dictionary, and IL tables into a record per track form which can be written onto disk for later retrieval by Pass 2.

The Recover routine, which attempts to recover from a source program syntax error in order that syntax checking can continue, should be revised to allow for the structure of BLISS-360.

A variety of optional debugging tools for use during compiler development and as an aid to source program checking should be developed. This could include traces of the IL production, BNF reductions, subroutines executed, as well as dumps of the IL, constant and dictionary tables.

A method should be developed for interpreting compile-time control parameters. This must include a form for control statements (eg., $LIST, and $DEBUG), a set of logical switches for retaining this information, and routines to execute the functions.

These parameters will take the place of the BLISS parameters and switches discussed in Ref. 2.

B. PASS 2

Since the XPL System was designed as a one-pass compiler, there is no structure available for the Pass 2 program. Therefore it must be designed completely to meet the requirements discussed below. In order

to understand some of these requirements, it will be necessary to have read the BLISS language description in Ref. 2.

1. Machine Language Production

Routines must be developed to read the IL tables and produce a form of S/360 ML which executes independently of any monitor system. As part of this function these routines must complete the implementation of the BLISS-360 structure access mechanism. This includes substitution of incarnation actual parameters for incarnation formal parameters in the structure size and structure access expressions, and inserting the IL for structure access expressions into the main IL sequence.

Some local code optimization capability should also be built in here.

2. Storage Allocation

Storage areas for constants and variables must be assigned in this pass. This includes such operations as computing the values of structure size expressions, assigning fixed locations for global and own variables and constants, and generating dynamic allocation mechanisms to allow for recursion and the reuse of local storage.

The structure used in BLISS-10 [Ref. 2] for both fixed and dynamic areas appears to be equally suitable for BLISS-360. It may be desirable, however, to split the fixed storage into several areas to allow these items to be stored nearer to their point of declaration. This could cost some memory space, for example, by having duplicate constant values, but should reduce the number of page faults in a time-shared environment by improving data contiguity.

3. Program Output

The output format for the object program must be designed to allow for linkage with other program modules. Communication links must be established for global and external routines and variables.

Initially, these links should be compatible with the requirements of the Linkage Editor routines of OS/360 in order to facilitate testing. However, the design should be sufficiently flexible that it can be changed easily to interface with other systems.

IV. IMPLEMENTING THE SYSTEM - PASS 1

The modifications to the Skeleton Compiler for BLISS-360 which have been completed are presented in detail in this section. In addition, a discussion of intermediate language, dictionary, and constant table production is presented, although these have generally not yet been programmed.

Excerpts from the Skeleton containing the completed BLISS-360 changes are shown in Appendix C. Basic to the understanding of Section IV is a knowledge of the data structures and procedures of the Skeleton [Ref. 3] and a familiarity with BLISS-10 [Ref. 2].

A. SYNTAX ANALYSIS

The modifications which are discussed below allow the Skeleton to perform a complete syntax check of a BLISS-360 program based upon the BNF in Appendix A. Some related additions, such as a diagnostic syntax trace, are shown here also for convenience.

1. Scanning

The Scan routine is responsible for extracting the next terminal symbol from the source program, and for setting the variable TOKEN to a value indicating the type of symbol found. Its structure remains as in the original Skeleton, although the individual cases for the terminal symbols have largely been rewritten.

The function of each of these cases is listed in Table I.

TABLE I.

Scan Routine Cases

Case	Function
0	detect illegal character, print error message and bypass it
1	bypass blanks
2	extract character strings and insert them into the constant table
3	not used
4	identify reserved words and identifiers, store and retrieve macros
5	extract numerical constants, convert them to internal format and store them in the constant table - conversion occurs in Scan for hexadecimals and in Consvert for all other types (ie. half word and full word integer, real, long real and packed decimal) -- only hexadecimals and integers can be converted at present
6	bypass the two forms of BLISS-360 comments: strings of characters bracketed either by %...% or by ?...end of a line
7	identify special characters

Macro manipulation is a special process of case 4. If the word MACRO is recognized, a call is made to the Putmac routine. This routine, which presently bypasses macros, will place the macro in the table defined in Table II.

Macros are placed in a special table instead of the combined symbol table and dictionary primarily to save space. The dictionary has many elements in each entry, most of which would not be needed for macros. Furthermore, the dictionary is designed to save information for Pass 2 whereas macro processing is complete in Pass 1.

An example showing the construction of a macro table is given in Fig. 3.

Macro retrieval is accomplished in case 4 by comparing each identifier name to the MACNAME entries to determine whether it is a macro call. If a match is made, the Getmac routine will be called.

This routine will be required to build a table of actual parameters for insertion into the macro body in MACSTR. It will then start scanning this MACSTR entry as if it were any other input string, except that when an ! is encountered, the appropriately numbered actual parameter is inserted.

One problem with this technique is that nested macro calls are allowed in BLISS-360. This will require Getmac to provide for the stacking of parameter lists and the MACSTR entries being scanned.

Once the process of storing or retrieving a macro is completed, Scan is reentered in order to retrieve the next symbol.

TABLE II.

Macro Table Definitions

MACTOP Bit (16); /* Next available entry in the macro table */

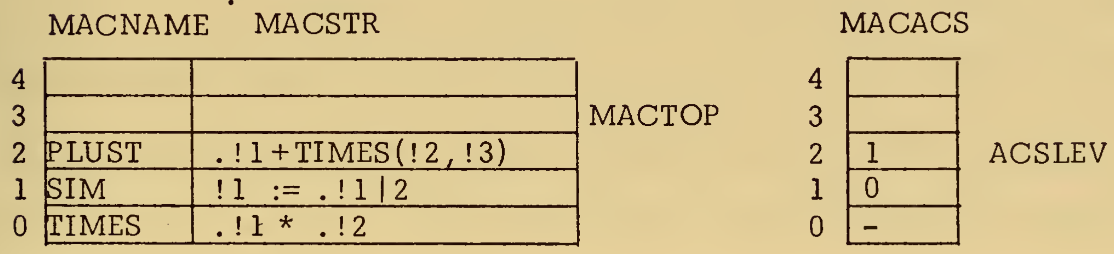
MACMAX Literally '100'; /* Maximum number of active macros */

MACNAME (MACMAX) Character; /* Contains the macro name. If the number of characters in the macro is greater than 256, it will contain a dummy entry, eg. M1, M2 for each additional block of 256 or fraction thereof. */

MACSTR (MACMAX) Character; /* The macro associated with the name in the same entry of MACNAME. As the macro is copied into MACSTR formal parameters from the macro's namelist will be replaced by ! number where number is the order of that parameter in the list.*/

MACACS (DICACSMAX) Bit (16); /* Defines the scope of macro definitions. The entry specified by block level pointer ACSLEV contains the macro table entry number of the first entry at the current block level.*/

```
BEGIN
        MACRO TIMES (X,Y) = .X * .Y$,
                  SIM (I) = I :=.I | 2 $;
                  .
                  .
                  .
        BEGIN
           MACRO PLUST (A,B,C) = .A + TIMES (B,C) $:
                  .
                  .
                  .
```



	MACNAME	MACSTR
4		
3		
2	PLUST	.!1+TIMES(!2,!3)
1	SIM	!1 := .!1 | 2
0	TIMES	.!1 * .!2

	MACACS
4	
3	
2	1
1	0
0	–

Fig. 3. Macro Storage Structure

2. Initialization

Changes to the Initialization routine are minor. They include primarily setting some new special indicators from the V table (a table containing all the terminal and non-terminal symbols in BLISS-360) and adding some CHARTYPE values to accommodate BLISS-360 comments and constants.

3. Recover

In the event of a source program syntax error the Recover routine attempts to find a meaningful place to allow the error to be bypassed and syntax checking to continue. The routine provided in Skeleton allows more to be bypassed than is necessary.

The Recover routine was, therefore, replaced with the routine used in Algol-E [Ref. 8] , modified to accommodate some additional terminal symbols of BLISS-360. Some further changes are still desirable, however, as a key place for the resumption of checking is immediately following a semi-colon, but the present version of BLISS-360 does not allow a semi-colon to follow the last expression in the block.

4. Debugging Aids

First attempts to syntax check a test program with the BLISS-360 modified Skeleton made it apparent that some simple debugging tools would be extremely useful. Thus, statements were inserted in many routines to print the name of the location just entered whenever the key item TESTIT has the value 1.

Another aid, which should be particularly useful when Synthesize is being programmed, is a syntax production trace. Just before each production is reduced, the Prodtrace routine is called to print the number of the production and its BNF form.

B. DATA TABLE DESIGN

Constant and variable data element information must be available for Pass 1 and saved for Pass 2. This is accomplished by the creation of a constant table (referred to as CONTAB) and a dictionary table (referred to as DICT).

The lengths of the vectors comprising each table were chosen arbitrarily and probably bear little relationship to what will be needed eventually. Note, however, that the use of registers by the XPL compiler (XCOM) places a limit on the total combined size of all tables.

1. Constant Table

Constant values are entered in this table from either the Scan or the Consvert routine. No attempt has been made to eliminate duplicate values. If desired, duplication could be removed in Pass 2 when constant storage assignment is performed.

The format for the constant table is given in Table III.

A plit is a special form of a constant [Ref. 1]. It is actually a sequence of constants which may be defined at compile time or load time, or may be another plit (referred to as a subplit).

TABLE III

Constant Table CONTAB and Related Control Variables

CONMAX Literally '200'; /* Maximum number of constants */

STRMAX Literally '1000'; /* Maximum number of characters in all strings plus digits/2 in all packed decimal constants */

CTYPE (CONMAX) Bit (8); /* Data type for this entry: 1-hexadecimal, 2-integer, 3- half word, 4-real, 5-long real, 6-string, 7-packed decimal, 8-plit */

VALPTR (CONMAX) Bit (16); /* Starting byte in CHARST for a string or packed decimal, or the word in NUMVAL containing the value of a hexadecimal, integer or real, or the first of two words in NUMVAL containing the value of a long real */

CHARST (STRMAX) Bit (8); /* A sequence of entries defined by NUMPTR and VALPTR forms a character string or a packed decimal number */

NUMVAL (CONMAX) Fixed; /* As defined by VALPTR each entry, or pair of entries for long real, contains an internal format numerical value */

PLTPT (CONMAX) Bit (16); /* Entry number in CONTAB or in DICT of the next entry in a plit chain */

PLTIND (CONMAX) Bit (8); /* Plit status of this entry: 0-not part of a plit, 1-PLTPT points to a CONTAB entry, 2-same as 1, but this is the last entry of a subplit, 3-PLTPT points to a DICT entry, 4-same as 3, but this is the last entry of a subplit, 5-end of a plit */

CONST Bit (16); /* Latest entry in CONTAB */

CHARPTR Bit (16); /* Next available entry in the CHARST vector */

NUMVALPTR Bit (16); /* Next available entry in the NUMVAL vector */

CONVTEMP Character; /* Temporarily holds a number in external format for conversion by Consvert */

PLTLEV Bit (8); /* Level of nesting in a plit */

PLTLIST Bit (16); /* The most recent entry in the plit list */

PLTLOC Bit (1); /* Table referred to by PLTLIST: 0-CONTAB, 1-DICT */

In order that the sequence comprising a plit be connected for proper storage allocation in Pass 2, its entries form a list with elements joined by PLTPT in CONTAB and DPLTPT in DICT. The head of the list will be in a special CONTAB entry. In order to control the plit list, the variables PLTLEV, PLTLIST and PLTLOC [Table III] are needed.

An example showing a typical plit is given in Fig. 4. Only the DICT and CONTAB variables pertaining to the structure are shown. An example showing CONTAB without plits is presented in Sec. V.A.

2. Dictionary

Variable data elements are entered into this table as the appropriate form is recognized in Synthesize. DICT is a combined data definition and symbol table, with the entire table being used in Pass 1 and only the data definition saved for Pass 2. In Pass 1, names are entered and looked up through a hash table. In Pass 2 DICT entries are accessed directly through operand entries in the IL. The DICT format is shown in Table IV.

Access to block levels in DICT is maintained through a table of pointers to the first entry of each active block. When a block is entered the number of the next available DICT entry is placed at the top of the access pointer table. Entries for the current block are then added to DICT with corresponding names going into the DNAME stack. As a block is ended, the access table entry for that block is removed and hash table entries referring to that block are reset to the location of the next previous entry in the chain, or to zero if the entry referred to was the end

```
BIND Y = PLIT (A,PLIT(B,C), PLIT 3H,'DE5K3' ,13);
```

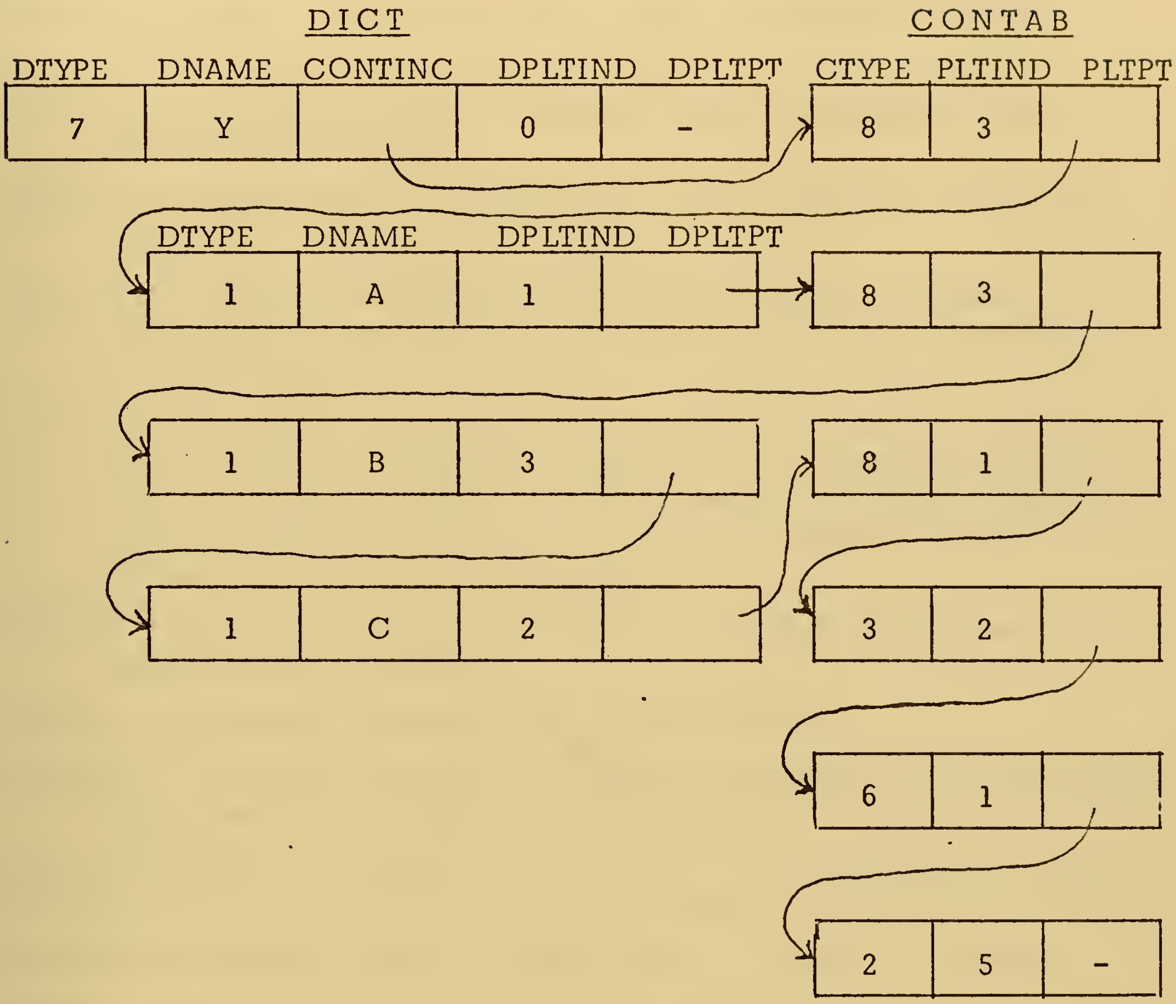


Fig. 4. Plit Storage Structure

TABLE IV.

Dictionary Table DICT and Related Access Control Variables

DICTMAX Literally '500'; /* Maximum number of definition entries in DICT */

DNAMEMAX Literally '150'; /* Maximum number of symbol entries active at any time */

DNSAVEMAX Literally '50'; /* Maximum number of external and global names */

DTYPE (DICTMAX) Bit (8); /* Data type for this entry: 1-variable, 2-formal parameter, 3-function, 4-machop, 5-map, 6-structure, 7-bind, 8-label */

SUBTP (DICTMAX) Bit (8); /* Subtype, for DTYPE of variable: 1-global, 2-own, 3-local, 4-register, 5-fpregister, 6-external; function: 1-local routine, 2-global routine, 3-external routine, 4-function, 5-forward function, 6-module; structure: number of STRUCTIL entries for the access expression */

DNSAVE (DNSAVEMAX) Character; /* Names saved for global and external variables and routines */

DNAME (DNAMEMAX) Character; /* Symbol name */

DNAMELOC (DICTMAX) Bit (16); /* DNAME entry number for active blocks, DNSAVE entry number for inactive blocks */

PBACK (DICTMAX) Bit (16); /* Next previous DICT entry with the same hash code, set to zero for the last entry in a chain */

HCODE (DICTMAX) Bit (16); /* Hash code for the name in DNAME */

STRUCT (DICTMAX) Bit (16); /* For DTYPE of variable, bind or map: the entry in DICT which defines the related structure, 0 for the implied vector structure; structure: the first entry in STRUCTIL for this access expression */

ILENT (DICTMAX) Bit (16); /* For DTYPE of bind: the IL entry specifying the value which is bound to the variable; label: the first IL entry of the expression to which it refers; map: the DICT entry of the variable being mapped; structure: the first entry in STRUCTIL for the size expression; 0 if size expression is not specified; register type variable: the register number specified, -1 if a specific register is not designated; any function type except external: the first IL entry of the first expression in the function */

TABLE IV (Continued)

```
CONTINC (DICTMAX) Bit (16); /* Entry into CONTAB for a DTYPE of
        machop; variable, bind or map: first entry in the INCACT vector,
        0 if no incarnation actuals specified */

WDALIGN (DICTMAX) Bit (8); /* Word boundary alignment: 0-don't care,
        1-half word, 2-full word, 3-double word; for DTYPE of structure:
        the number of STRUCTIL entries for the size expression */

DPLTPT (DICTMAX) Bit (16); /* Has the same meaning as PLTPT in
        CONTAB */

DPLTIND (DICTMAX) Bit (8); /* Has the same meaning as PLTIND in
        CONTAB */
NUMINC (DICTMAX)Bit (8); /* For DTYPE of variable, bind or map: the
        number of entries in INCACT */

INCMAX Literally '300'; /* Maximum number of incarnation actuals*/

INCACT (INCMAX) Fixed; /* Incarnation actual values */

DICACSMAX  Literally '100'; /* Maximum number of block levels */

DICACS (DICACSMAX) Bit (16); /* Access level pointers for active DICT
        blocks */

ACSLEV Bit (16); /* Current access level entry in DICACS */

DICTOP Bit (16); /* Next available entry in DICT */

DNAMECTR Bit (16); /* Most  recent entry in DNAME */

DNSAVECTR Bit (16); /* Most recent entry in DNSAVE */

INCTOP Bit (16); /* Next available entry in INCACT */

HASHT (251) Bit (16); /* Hash table for inserting and locating DICT
        entries */
```

of a chain. Note, however, that DICTOP is not reduced, thus allowing the data to be preserved for transmission to Pass 2 although the entries have been removed from consideration in Pass 1.

An additional block end function is the moving of global and external names to the DNSAVE vector with a corresponding resetting of DNAMELOC to point to the new location. The DNAMECTR will then be decremented by the number of entries in the block just ended and its corresponding entries in DNAME will be set to the null string.

This scheme for handling names is used partly to save space, but primarily to accommodate a restriction of the XPL Compiler Generator System, which allows only a limited number of character descriptors.

The variables which define the data elements must be saved for Pass 2. These variables are DTYPE, SUBTP, STRUCT, ILENT, CONTAB, WDALIGN, INCACT, DPLTPT and DPLTIND, shown in Table IV. DNAMELOC and DNSAVE must also be saved in order to provide essential interface information.

An example showing the structure of DICT and its related access vectors is given in Sec. V.A.

C. INTERMEDIATE LANGUAGE PRODUCTION

Generation of the object program in an intermediate language (IL) form is one of the primary tasks of Pass 1. In this section the IL will be defined and some discussion concerning its use for some unique BLISS-360 cases will be presented.

1. Possible Forms of IL

Several forms for the internal representation of a program are discussed by Gries [Ref. 9]. Three of these forms were considered to be potentially acceptable: quadruples, triples, and indirect triples.

The quadruples form of IL has four basic elements: operator, operand 1, operand 2 and results. The operator defines the operational relationship between operand 1 and operand 2. Results defines the location for storing the result of this operation. Each entry has a place for four elements although they are not all meaningful for some operations.

Triples have a form that is identical to that for quadruples except that there is no results field. Thus, when results from a previous IL entry are required the operand field will refer directly to that entry.

Indirect triples are a modification of the triples form that lends itself more readily to optimization. In this form, one table defines the order in which operations occur. The operations in the form of triples are in a second table. When a new triple is formed it is compared to the triples table. If it exactly matches an existing entry, the current entry of the operations table is set to that entry number. Otherwise, a new triples table entry is created with the current operations table entry set to the new triples entry number.

2. Design of the IL Table

The triples form was selected as the most suitable IL for a first implementation. It is simpler than quadruples in that it does not

require generating and keeping track of large numbers of temporary result registers. It also requires somewhat less storage space.

Indirect triples are a potentially more useful form, although they would be more complex and time-consuming to generate. Furthermore, if code optimization is desired at a later date, switching from the triples form would require relatively minor changes.

An example of the use of IL is given in Sec. V.A.

a. IL and Its Operators

The format of the IL table is given in Table V.

Some operation codes which will be used in OPCODE are shown with their meanings in Table VI. The codes shown should be sufficient to represent a usable subset of BLISS-360, although more will be required to represent the entire language.

Each code name is declared as a bit (8) variable which is assigned a numeric value by the Initialization routine in the order in which it is defined. Using the same definitions and initialization in Pass 2 will avoid conflicts when codes are added or deleted.

b. IL for Structure Access

BLISS-360 has a unique form of data structure access [Refs. 1, 2, 10, and 11] which creates some special requirements for IL generation. Structure size and structure access expressions are not executed at the position in the program structure where they are declared. In addition, the definition of any variables in these expressions must be

TABLE V.

Intermediate Language Table IL

```
NUMBOPS Literally '46'; /* Number of IL operators */

ILMAX Literally '1000'; /* Maximum size of a segment of IL.  (The IL
          table will be generated and written in segments with the breaks
          coming after function or block ends.) */

OPCODE (ILMAX) Bit (8); /* Operation code */

OP1 (ILMAX) Bit (16); /* Operand one.  If OPTYPE1 = 1, it refers to a
          DICT entry, =2 a CONTAB entry, =3 another IL entry, =4 a temporary
          value table entry */

OP2 (ILMAX) Bit (16); /* Operand two.  If OPTYPE2 = 1, it refers to a
          DICT entry, =2 a CONTAB entry, =3 another IL entry, =4 a
          temporary value table entry */

OPTYPE1 (ILMAX) Bit (8); /* Type of OP1: 0-illegal, 1-variable,
          2-constant, 3-IL entry, 4-temporary */

OPTYPE2 (ILMAX) Bit (8); /* Type of OP2: 0-illegal, 1 - variable,
          2-constant, 3-IL entry, 4-temporary */

OPDOTS1 (ILMAX) Bit (8); /* Level of indirect references for OP1 */

OPDOTS2 (ILMAX) Bit (8); /* Level of indirect references for OP2 */
```

TABLE VI.

Operation Codes for IL

Operation Code	Meaning	
ADDX, ADDP, ADDH, ADDE, ADDL	OP1 + OP2	the X suffix signifies integer arithmetic, P - packed decimal, H - half word, E - real, L - long real
SUBX, SUBP, SUBH, SUBE, SUBL	OP1 - OP2	
MULX, MULP, MULH, MULE, MULL	OP1 * OP2	
DIVX, DIVP, DIVE, DIVL	OP1 / OP2	
ASSG	OP1 := OP2	
UMIN	-OP1 (unary minus)	
BLCK	block entry OP1-block level, OP2-first DICT entry for the block	
BLKE	block exit OP1-block level	
TEST	compare OP1:OP2	
BLSS	branch on <	branch on the results of test, OP1 - IL entry of the test OP2 - IL entry branched to if the condition is true
BLEQ	branch on ≤	
BEQL	branch on =	
BNEQ	branch on ≠	
BGEQ	branch on ≥	
BGTR	branch on >	
BFLS	branch on false	OP1 is tested for 0(F) or 1(T) in the least significant bit, OP2 - branch location
BTRU	branch on true	
BUNC	unconditional branch OP1 - branch location, OP2 - location of exit value for leave	
SHFT	OP1 | OP2	
EQVL	OP1 eqv OP2	
EXOR	OP1 xor OP2	

TABLE VI (Continued)

MODU	OP1 mod OP2
ORIT	OP1 or OP2
ANDT	OP1 and OP2
KNOT	not OP1
FCTT	function or routine declaration, OP1 - first related DICT entry
FCTE	end of function declaration
FUNC	function call, OP1 - DICT entry of the function, OP2 - DICT, IL or CONTAB entry of an actual parameter value, OPDOTS1 - order of this parameter in the formal parameter list
FUNE	function call end - has the same format as FUNC usage: for each actual parameter expression except the last there will be one entry with the FUNC operator. The last parameter will be represented by a FUNE operator entry.
RTRN	return OP1 - entry of escape value expression, OP2 - DICT entry of related function
STHD	structure access, OP1 - DICT entry of variable being accessed, OP2 - access actual value
STAC	structure access end - has the same format as STHD usage: for each access actual expression except the last there will be one entry with the STHD operator. The last access actual will be represented by a STAC operator.

those in effect at the point of the structure declaration, not those in effect at the time of execution.

These expressions, therefore, must be syntactically and semantically analyzed when they are declared, then saved for insertion where needed to compute a structure storage size or to access a variable.

A method for saving the structure expressions is to generate a special IL table (STRUCTIL) for them. The form of STRUCTIL is exactly that of IL except that it has the additional variables POSIND1 and POSIND2. These variables indicate the position of operands OP1 and OP2, respectively, in the formal parameter list of the structure declaration. If either of the position indicators is zero, the corresponding operand is not a parameter. If either one is not zero, then the corresponding indirect reference variable (OPDOTS1 or OPDOTS2) has an additional meaning. This meaning is that if the OPDOTS variable is zero, then the corresponding operand is an incarnation formal. If it is not zero, the corresponding operand is an access formal.

V. PROGRAM EXAMPLES

Sample BLISS-360 programs are presented here to illustrate the implementation features of Sec. IV.

A. TABLE DESIGN EXAMPLE

In this section the DICT, CONTAB and IL tables are shown for the program in Fig. 5. The DICT table, and its related tables HASHT, DNAME, DNSAVE, DICACS and INCACT are shown as they would appear when the program had been analyzed to line 6 in Table VII and again after it had been analyzed to line 10 in Table VIII. This serves to illustrate the effects of closing one block, then opening another.

IL, STRUCTIL and CONTAB are shown as they would appear upon completion of Pass 1 of the compiler in Tables IX, X, and XI.

The DICT and CONTAB variables associated with plits are not shown since they are not pertinent to the example.

Whenever a dash appears for a variable's value that variable has no meaning for that DICT entry.

No algorithm was used to compute entry values in the hash table HASHT. Numbers were chosen arbitrarily for illustrative purposes.

B. SYNTAX ANALYSIS EXAMPLES

Two additional examples of BLISS-360 programs are shown in Appendix B. These programs were each syntactically analyzed

successfully by the BLISS-360 Skeleton [Appendix C]. Program 2 was also run with synthesis tracing turned on. Partial results of that run are also presented in Appendix B.

```
1    MODULE =
2      BEGIN LOCAL AA, AB <4> ;
3        STRUCTURE VEC2 <P,Q> = <P+Q> (.VEC2 + .P + .Q-Q);
4        AA :=.AA+1;
5      BEGIN GLOBAL VEC2  C1: C2  <5,7> ;  LOCAL C3; REGISTER RX=#B;
6        IF .C3 THEN  C1 <2,9> :=..C3/RX
7      END
8      BEGIN OWN DD;  LOCAL KD <2>;
9        BIND X = 5;
10         AA :='A CHAR STRING';
11         DD | X :=.DD MOD X;
12         KD<X + 1>  -  .DD
13     END
14   END  ELUDOM  EOF
```

Fig. 5

BLISS-360 Program Example

TABLE VII. DICT Entries at Line 6, Fig. 5

	DTYPE	SUBTP	STRUCT	ILENT	CONTINC	NUMINC	WDALIGN	PBACK	HCODE	DNAMELOC	
13											
12											
11											
10											← DICTOP
9	1	4	0	11	0	-	0	0	8	8	
8	1	3	0	-	0	-	0	6	5	7	
7	1	1	3	-	2	2	0	4	11	6	
6	1	1	3	-	2	2	0	0	5	5	
5	2	-	0	-	0	-	0	0	14	4	
4	2	-	0	-	0	-	0	2	11	3	
3	6	3	2	1	0	-	1	0	2	2	
2	1	3	0	-	1	1	0	0	11	1	
1	1	3	0	-	0	-	0	0	6	0	
0	-	-	-	-	-	-	-	-	-	-	

	HASHT
16	
15	
14	5
13	
12	
11	7
10	
9	
8	9
7	
6	1
5	8
4	
3	
2	3
1	
0	

	DNAME	
9		
8	RX	← DNAME CTR
7	C3	
6	C2	
5	C1	
4	Q	
3	P	
2	VEC2	
1	AB	
0	AA	

	DICACS	
5		
4		
3		
2	6	← ACSLEV
1	1	
0		

	INCACT	
5		
4		← INCTOP
3	7	
2	5	
1	4	
0	-	

	DNSAVE
3	
2	
1	
0	

← DNSAVECTR

TABLE VIII. DICT Entries at Line 10, Fig. 5

	PBACK	HCODE	DNAMELOC	STRUCT	ILENT	CONTINC	NUMINC	DTYPE	SUBTP	WDALIGN	
13											←DICTOP
12	0	13	7	0	9	0	–	7	–	0	
11	0	5	6	0	–	4	1	1	3	0	
10	4	11	5	0	–	0	–	1	2	0	
9	–	–	–	0	11	0	–	1	4	0	
8	–	–	–	0	–	0	–	1	3	0	
7	–	–	1	3	–	2	2	1	1	0	
6	–	–	0	3	–	2	2	1	1	0	
5	0	14	4	0	–	0	–	2	–	0	
4	2	11	3	0	–	0	–	2	–	0	
3	0	2	2	0	3	0	–	6	3	1	
2	0	11	1	0	–	1	1	1	3	0	
1	0	6	0	0	–	0	–	1	3	0	
0	–	–	–	–	–	–	–	–	–	–	

	HASHT
16	
15	
14	5
13	12
12	
11	10
10	
9	
8	
7	
6	1
5	11
4	
3	
2	3
1	
0	

	DNAME	
9		
8		
7	X	←DNAME CTR
6	KD	
5	DD	
4	Q	
3	P	
2	VEC2	
1	BB	
0	AA	

	DICACS	
5		
4		
3		
2	10	←ACSLEV
1	1	
0		

	INCACT	
6		
5		←INCTOP
4	2	
3	7	
2	5	
1	4	
0	–	

	DNSAVE	
3		
2		
1	C2	←DNSAVECTR
0	C1	

TABLE IX. IL Entries for Fig. 5

	OPCODE	OP1	OPTYPE 1	OPDOTS 1	OP2	OPTYPE 2	OPDOTS 2
1	BLCK	1	-	-	1	-	-
2	ADDX	1	D	1	1	C	-
3	ASSG	1	D	0	2	I	0
4	BLCK	2	-	-	6	-	-
5	BFLS	8	D	1	10	I	-
6	STHD	6	D	0	2	C	-
7	STAC	6	D	0	3	C	-
8	DIVX	8	D	2	9	D	0
9	ASSG	7	I	0	8	I	0
10	BLKE	2	-	-	-	-	-
11	BLCK	2	-	-	10	-	-
12	ASSG	12	D	0	8	C	-
13	ASSG	1	D	0	9	C	-
14	SHFT	10	D	0	12	D	0
15	MODU	10	D	1	12	D	0
16	ASSG	14	I	0	15	I	0
17	ADDX	12	D	0	10	C	-
18	STAC	11	D	0	17	I	0
19	SUBX	18	I	0	10	D	1
20	BLKE	2	-	-	-	-	-
21	BLKE	1	-	-	-	-	-

References to DICT - D, CONTAB - C, IL - I

TABLE X. STRUCTIL Entries for Fig. 5

	OPCODE	OP1	OPTYPE 1	OPDOTS 1	POSIND 1	OP2	OPTYPE 2	OPDOTS 2	POSIND 2
1	ADDX	4	D	0	1	5	D	0	2
2	ADDX	3	D	1	0	4	D	1	1
3	ADDX	2	I	0	0	5	D	1	2
4	SUBX	3	I	0	0	5	D	0	2

TABLE XI. CONTAB Entries for Fig. 5

	CTYPE	NUMPTR	VALPTR	
11				
10	2	–	9	←CONST
9	6	13	0	
8	2	–	8	
7	2	–	7	
6	2	–	6	
5	2	–	5	
4	1	–	4	
3	2	–	3	
2	2	–	2	
1	2	–	1	
0	2	–	0	

	NUMVAL	
13		
12		
11		
10		←NUMVALPTR
9	1	
8	5	
7	2	
6	9	
5	2	
4	11	
3	7	
2	5	
1	1	
0	4	

	CHARST	
15		
14		
13		←CHARPTR
12	G	
11	N	
10	I	
9	R	
8	T	
7	S	
5		
5	R	
4	A	
3	H	
2	C	
1		
0	A	

VI. CONCLUSIONS

Designing and writing a compiler for a language with the complexity and versatility of BLISS-360 is a large undertaking. The work described in this thesis has provided a beginning for this task.

A. WHAT HAS BEEN ACCOMPLISHED

An organized approach on which to base continuing activity on the compiler has been presented in this thesis.

A two-pass compiler structure was designed based upon the XPL Compiler Generating System [Ref. 2] . The first of these passes, which is based on the XPL prototype compiler Skeleton, was examined in some detail. Key tables were designed to be added to this program. In addition, the Skeleton was modified to perform a syntax analysis of BLISS-360 programs. General requirements were defined for the functions to be performed by the second pass.

B. WHAT REMAINS TO BE DONE

Producing a working BLISS-360 compiler from the foundation provided in this thesis is the task ahead.

Intermediate language production is the largest single task yet to be done in Pass 1. At present, the BNF [Appendix A] contains 238 productions, each of which must be examined to see what IL entries, if any, must be created for it. Code must then be constructed to produce

these entries. A number of other functions, such as dictionary entry and lookup, and the evaluation of compile-time constant expressions, must also be programmed.

Pass 2 must be designed in detail from the general requirements set forth in Sec. III.B, and programmed. This will require a working knowledge of S/360 machine language and of the OS/360 interface formats. In addition, some understanding of optimization techniques, such as those discussed in Ref. 2 and 9, would be very useful.

APPENDIX A

BNF Description of BLISS-360

```
 1   <MODULE>  ::=  <MODULE HEAD> <MODULE END>

 2   <MODULE HEAD>  ::=  <HEADING> <EQE>
 3                     |  <HEADING> <ID CLAUSE>

 4   <HEADING>  ::=  MODULE

 5   <MODULE END>  ::=  ELUDOM

 6   <E>  ::=  <LABELLED EXPR>
 7          |  <UNLABELLED EXPR>

 8   <LABELLED EXPR>  ::=  <LABEL> : <UNLABELLED EXPR>

 9   <UNLABELLED EXPR>  ::=  <BALANCED EXPRESSION>
10                       |  <UNBALANCED EXPRESSION>

11   <BALANCED EXPRESSION>  ::=  <SIMPLE EXPRESSION>
12                           |  <CONTROL EXPRESSION>
13                           |  <TEST TRUE PART> <BAL PART>

14   <UNBALANCED EXPRESSION>  ::=  <IF CLAUSE> <FIN THEN CLAUSE>
15                             |  <TEST TRUE PART> <UNBAL PART>

16   <IF CLAUSE>  ::=  IF <E>

17   <TEST TRUE PART>  ::=  <IF CLAUSE> <THEN CLAUSE>

18   <THEN CLAUSE>  ::=  <THEN> <BALANCED EXPRESSION>

19   <BAL PART>  ::=  ELSE <BALANCED EXPRESSION>

20   <UNBAL PART>  ::=  ELSE <UNBALANCED EXPRESSION>

21   <SIMPLE EXPRESSION>  ::=  <P11>
22                         |  <P11> <RIGHT PART>

23   <RIGHT PART>  ::=  := <E>

24   <P11>  ::=  <P10>
25            |  <P11> XOR <P10>
26            |  <P11> EQV <P10>
```

```
27   <P10>   ::=  <P9>
28             |  <P10> OR <P9>

29   <P9>   ::=  <P8>
30            |  <P9> AND <P8>

31   <P8>   ::=  <P7>
32            |  NOT <P7>

33   <P7>   ::=  <P6>
34            |  <P6> <RELATION> <P6>

35   <P6>   ::=  <P5>
36            |  - <P5>
37            |  <P6> + <P5>
38            |  <P6> - <P5>

39   <P5>   ::=  <P4>
40            |  <P5> * <P4>
41            |  <P5> / <P4>
42            |  <P5> MOD <P4>

43   <P4>   ::=  <P3>
44            |  <P4> | <P3>

45   <P3>   ::=  <P1>
46            |  . <P3>

47   <P1>   ::=  <LITERAL>
48            |  <STRUCTURE ACCESS>
49            |  <FUNCTION CALL>
50            |  <BLOCK>

51   <RELATION>  ::=  EQL
52                  |  NEQ
53                  |  LSS
54                  |  LEQ
55                  |  GTR
56                  |  GEQ

57   <LITERAL>  ::=  <IDENTIFIER>
58                |  <NUMBER>
59                |  <STR>

60   <STR>   ::=  <STRING>
61             |  <STRING TYPE> <STRING>
```

```
62   <STRING TYPE>  ::=  ASCII

63   <STRUCTURE ACCESS>  ::=  <STRUCTURE ACCESS HEAD> <E> >

64   <STRUCTURE ACCESS HEAD>  ::=  <IDENTIFIER> <
65                              |  <STRUCTURE ACCESS HEAD> <E> ,

66   <FUNCTION CALL>  ::=  <FUNCTION HEAD> <EPAR>
67                      |  <FUNCTION HEAD> )

68   <FUNCTION HEAD>  ::=  <IDENTIFIER> (
69                      |  <FUNCTION HEAD> <E> ,

70   <BLOCK>  ::=  <BLOCK HEAD> <BLOCK END>
71              |  <BL HEAD4> <BLOCK END>

72   <BLOCK HEAD>  ::=  <BL HEAD1> <DECLARATION>
73                   |  <BL HEAD2> <E>

74   <BL HEAD1>  ::=  <BL HEAD4>
75                 |  <BL HEAD1> <DECLARATION> ;

76   <BL HEAD2>  ::=  <BL HEAD1>
77                 |  <BL HEAD2> <E> ;

78   <BL HEAD4>  ::=  BEGIN
79                 |  (

80   <BLOCK END>  ::=  END
81                  |  )

82   <CONTROL EXPRESSION>  ::=  <LOOP EXPRESSION>
83                           |  <ESCAPE EXPRESSION>
84                           |  <CHOICE EXPRESSION>
85                           |  <COROUTINE EXPRESSION>

86   <LOOP EXPRESSION>  ::=  <WHILE EXPRESSION>
87                        |  <UNTIL EXPRESSION>
88                        |  <DO WHILE EXPRESSION>
89                        |  <DO UNTIL EXPRESSION>
90                        |  <INCREMENT EXPRESSION>
91                        |  <DECREMENT EXPRESSION>

92   <WHILE EXPRESSION>  ::=  <WHILE CLAUSE> <DO CLAUSE>

93   <WHILE CLAUSE>  ::=  WHILE <E>
```

```
 94   <DO CLAUSE>  ::=  DO <E>
 95   <UNTIL EXPRESSION>  ::=  <UNTIL CLAUSE> <DO CLAUSE>
 96   <UNTIL CLAUSE>  ::=  UNTIL <E>
 97   <DO WHILE EXPRESSION>  ::=  <DO CLAUSE> <WHILE CLAUSE>
 98   <DO UNTIL EXPRESSION>  ::=  <DO CLAUSE> <UNTIL CLAUSE>
 99   <INCREMENT EXPRESSION>  ::=  <INCR HEAD> <RIGHT ID PART>
100   <INCR HEAD>  ::=  INCR <IDENTIFIER>
101   <RIGHT ID PART>  ::=  <STEP EXPRESSION> <DO CLAUSE>
102   <STEP EXPRESSION>  ::=  <STEP PART> <ID PART>
103                        |  <STEP PART>
104   <STEP PART>  ::=  <START STEP> <END STEP>
105   <START STEP>  ::=  FROM <E>
106   <END STEP>  ::=  TO <E>
107   <ID PART>  ::=  BY <E>
108   <DECREMENT EXPRESSION>  ::=  <DECR HEAD> <RIGHT ID PART>
109   <DECR HEAD>  ::=  DECR <IDENTIFIER>
110   <ESCAPE EXPRESSION>  ::=  <ENVIRONMENT ESCAPE>
111                          |  <PROCEDURE ESCAPE>
112   <ENVIRONMENT ESCAPE>  ::=  <LOOP ESCAPE>
113                           |  <CONTROL ENVIRONMENT ESCAPE>
114   <LOOP ESCAPE>  ::=  <EXIT>
115                    |  <EXIT EXPRESSION>
116   <EXIT>  ::=  EXITLOOP
117   <EXIT EXPRESSION>  ::=  EXITLOOP <EXIT END>
118   <EXIT END>  ::=  <BRACED E>
119                  |  <ESCAPE VALUE>
120                  |  <BRACED E> <ESCAPE VALUE>
```

```
121   <BRACED E>  ::=  < <E> >

122   <ESCAPE VALUE>  ::=  <WITH CLAUSE>

123   <WITH CLAUSE>  ::=  WITH <UNLABELLED EXPR>

124   <CONTROL ENVIRONMENT ESCAPE>  ::=  <LABELLED LEAVE>
125                                    |  <LABELLED LEAVE> <ESCAPE VALUE>

126   <LABELLED LEAVE>  ::=  LEAVE <IDENTIFIER>

127   <PROCEDURE ESCAPE>  ::=  <RETURN>
128                         |  <RETURN EXPRESSION>

129   <RETURN>  ::=  RETURN

130   <RETURN EXPRESSION>  ::=  RETURN <ESCAPE VALUE>

131   <CHOICE EXPRESSION>  ::=  <CHOICE HEAD> <CHOICE END>

132   <CHOICE HEAD>  ::=  <LEFT CHOICE> <E> OF

133   <LEFT CHOICE>  ::=  CASE
134                   |  SELECT
135                   |  <LEFT CHOICE> <E> ,

136   <CHOICE END>  ::=  <CASE END>
137                  |  <SELECT END>

138   <CASE END>  ::=  <SET HEAD> TES
139                |  <SET HEAD> <E> TES

140   <SET HEAD>  ::=  SET
141                |  <SET HEAD> ;
142                |  <SET HEAD> <E> ;

143   <SELECT END>  ::=  <NSET HEAD> TESN
144                  |  <NSET HEAD> <NE> <E> TESN

145   <NSET HEAD>  ::=  NSET
146                 |  <NSET HEAD> <NE> <E> ;

147   <NE>  ::=  <XE>
148          |  ALWAYS :
149          |  OTHERWISE :

150   <XE>  ::=  <UNLABELLED EXPR> :
```

```
151   <COROUTINE EXPRESSION>   ::=  <CREATE EXPRESSION>
152                             |   <EXCHJ EXPRESSION>

153   <CREATE EXPRESSION>   ::=   <CREATE HEAD> <CREATE END>

154   <CREATE HEAD>   ::=   <CREATE> <FUNCTION CALL>

155   <CREATE>   ::=   CREATE

156   <CREATE END>   ::=   <LOC CLAUSE> <LENGTH CLAUSE> <FIN THEN CLAUSE>

157   <LOC CLAUSE>   ::=   AT <E>

158   <LENGTH CLAUSE>   ::=   LENGTH <E>

159   <FIN THEN CLAUSE>   ::=   <THEN> <UNLABELLED EXPR>

160   <THEN>   ::=   THEN

161   <EXCHJ EXPRESSION>   ::=   <EXCHANGE HEAD> <EPAR>

162   <EXCHANGE HEAD>   ::=   <EXCHANGE> <BL HEAD4>
163                      |    <EXCHANGE HEAD> <E> ,

164   <EXCHANGE>   ::=   EXCHJ

165   <EPAR>   ::=   <E> <BLOCK END>

166   <DECLARATION>   ::=   <TYPE DECLARATION>
167                    |    <ML DECLARATION>

168   <ML DECLARATION>   ::=   <SINGLE ML DECLARATION>
169                       |    <ALL ML DECLARATION>

170   <ALL ML DECLARATION>   ::=   ALLMACHOP

171   <SINGLE ML DECLARATION>   ::=   <ML HEAD> <ID CLAUSE>

172   <ML HEAD>   ::=   MACHOP
173                |    <ML HEAD> <ID CLAUSE> ,

174   <ID CLAUSE>   ::=   <IDENTIFIER> <EQE>

175   <EQE>   ::=   = <E>
```

```
176   <TYPE DECLARATION>  ::=  <FUNCTION DECLARATION>
177                          |  <STRUCTURE DECLARATION>
178                          |  <ALLOCATION DECLARATION>
179                          |  <BIND DECLARATION>
180                          |  <REGISTER DECLARATION>
181                          |  <LABEL DECLARATION>

182   <LABEL DECLARATION>  ::=  <LABEL HEAD> <IDENTIFIER>

183   <LABEL HEAD>  ::=  LABEL
184                  |  <LABEL HEAD> <IDENTIFIER> ,

185   <STRUCTURE DECLARATION>  ::=  <STRUCTURE HEAD> <EQE>
186                              |  <STRUCTURE HEAD> <STRUCTURE END>

187   <STRUCTURE HEAD>  ::=  <STRUCTURE NAME>
188                      |  <STRUCTURE NAME> <BRAC ID> <IDENTIFIER> >

189   <STRUCTURE NAME>  ::=  STRUCTURE <IDENTIFIER>

190   <BRAC ID>  ::=  <
191               |  <BRAC ID> <IDENTIFIER> ,

192   <STRUCTURE END>  ::=  = <BRACED E> <E>

193   <FUNCTION DECLARATION>  ::=  <TYPE FUNCTION>
194                             |  <FUNCTION TYPE LOC>

195   <TYPE FUNCTION>  ::=  <TYPE FUNCTION HEAD> <EQE>

196   <TYPE FUNCTION HEAD>  ::=  <PARAMLESS FUNCTION>
197                          |  <PARAM FUNCTION> <IDENTIFIER> )

198   <PARAM FUNCTION>  ::=  <PARAMLESS FUNCTION> (
199                      |  <PARAM FUNCTION> <IDENTIFIER> ,

200   <PARAMLESS FUNCTION>  ::=  <FUNCTION TYPE> <IDENTIFIER>

201   <FUNCTION TYPE>  ::=  FUNCTION
202                     |  ROUTINE
203                     |  GLOBALROUTINE

204   <FUNCTION TYPE LOC>  ::=  <TYPE LOC HEAD> <FUNCTION CALL>

205   <TYPE LOC HEAD>  ::=  EXTERNAL
206                     |  FORWARD
207                     |  <TYPE LOC HEAD> <FUNCTION CALL> ,
```

```
208   <ALLOCATION DECLARATION>  ::=  <ALLOC1>

209   <ALLOC1>  ::=  <ALLOC HEADING> <STRUCTURE ALLOC>
210              |   <ALLOC HEADING> <IDENTIFIER>
211              |   <ALLOC HEADING> <STRUCTURE ACCESS>
212              |   <ALLOC0> <IDENTIFIER>
213              |   <ALLOC0> <STRUCTURE ACCESS>

214   <ALLOC0>  ::=  <ALLOC1> :

215   <STRUCTURE ALLOC>  ::=  <IDENTIFIER> <IDENTIFIER>
216                       |   <IDENTIFIER> <STRUCTURE ACCESS>

217   <ALLOC HEADING>  ::=  <ALLOCATE HEAD>
218                     |   <BIND HEAD>

219   <ALLOCATE HEAD>  ::=  <ALLOCATE TYPE>
220                     |   <ALLOC1> ,

221   <ALLOCATE TYPE>  ::=  GLOBAL
222                     |   OWN
223                     |   LOCAL
224                     |   EXTALLOC
225                     |   REGALLOC
226                     |   FPREGALLOC
227                     |   MAP

228   <BIND HEAD>  ::=  <BIND HEADING>
229                 |   <ALLOC2> ,

230   <BIND HEADING>  ::=  BIND

231   <BIND DECLARATION>  ::=  <ALLOC2>

232   <ALLOC2>  ::=  <ALLOC1> <EQE>

233   <REGISTER DECLARATION>  ::=  <REGISTER HEAD> <REGISTER END>

234   <REGISTER HEAD>  ::=  REGISTER
235                     |   FPREGISTER
236                     |   <REGISTER HEAD> <REGISTER END> ,

237   <REGISTER END>  ::=  <ID CLAUSE>
238                    |   <STRUCTURE ACCESS> <EQE>
```

APPENDIX B

Sample BLISS-360 Programs

Program Example No. 1

```
  NPGS  -|-  BLISS-360  -|- -|- MOD 0        JUNE 2, 1972.  CLOCK TIME = 19:11:13.49.

TODAY IS JUNE 2, 1972.  CLOCK TIME = 19:12:12.44.

   1 |     %  TEST PROGRAM # 1   %
   2 |  MODULE  ABC  =
   3 |    BEGIN   LOCAL  L1:L2<10>, L3:L6;
   4 |      L1<2> := .L2<3>+.(L3*2);
   5 |     BEGIN  GLOBAL  XT ;  LOCAL L4;
   6 |       DECR XT FROM .L2<1> TO 4 BY 1 DO
   7 |       BEGIN
   8 |         IF .L3 OR .L4  THEN   ?  SAMPLE END OF LINE COMMENT
   9 |          BEGIN  LOCAL  L5;
  10 |           ←..L3*L4 / .L2<.L4>    END
  11 |       END;
  12 |      .L4|3;  %  A SPECIMEN MID-LINE COMMENT %  L3+4
  13 |     END;
  14 |     L3 + L6 := .L3/5 * .L6
  15 |    END
  16 |  ELUDOM
  17 |      EOF  EOF  EOF
END OF CHECKING JUNE 2, 1972.  CLOCK TIME = 19:12:14.92.

17 CARDS WERE CHECKED.
NO ERRORS WERE DETECTED.

TOTAL TIME IN CHECKER    0:0:3.34.
SET UP TIME              0:0:0.69.
ACTUAL CHECKING TIME     0:0:1.95.
CLEAN-UP TIME AT END     0:0:0.70.
CHECKING RATE: 523 CARDS PER MINUTE.
```

Program Example No. 2

```
  NPGS  -|-  BLISS-360  -|- -|- MOD 0        JUNE 2, 1972.  CLOCK TIME = 19:13:38.46.

TODAY IS JUNE 2, 1972.  CLOCK TIME = 19:14:41.05.

   1 |       %   TEST PROGRAM # 2    %
   2 |   MODULE =
   3 |      BEGIN  GLOBAL  AA<4>, AB<7>, AC ;
   4 |        STRUCTURE  VEC3 <I,J,K> = <I*J-K> .VEC3 + .I * J + (.J-3) + .K;
   5 |
   6 |        BEGIN  LOCAL  VEC3 CA<3,4,2>, CB:CC  ;
   7 |          ROUTINE CATZ =  ( LOCAL  DA:DB:DC;    ?  A NONSENSE FUNCTION
   8 |            DA:=.DB:=..DA;     .DC MOD .DB ) ;
   9 |          BIND Y = .CA<.CB, .CC, 2>;
  10 |           CA<1,2,3> :=  .CATZ() ;
  11 |           AA<.CB> := .CB XOR .CC
  12 |        END;
  13 |        AA<3> := IF NOT AC THEN AC|(-2) ELSE AC|3
  14 |     END
  15 |   ELUDOM     EOF
END OF CHECKING JUNE 2, 1972.  CLOCK TIME = 19:14:43.05.

15 CARDS WERE CHECKED.
NO ERRORS WERE DETECTED.

TOTAL TIME IN CHECKER      0:0:3.00.
SET UP TIME                0:0:0.60.
ACTUAL CHECKING TIME       0:0:1.60.
CLEAN-UP TIME AT END       0:0:0.80.
CHECKING RATE: 562 CARDS PER MINUTE.
```

Program Example No. 2 with Syntax Production Trace

```
  NPGS  -|-  BLISS-360  -|- -|- MOD 0         JUNE 14, 1972.  CLOCK TIME = 15:30:30.43.

TODAY IS JUNE 14, 1972.  CLOCK TIME = 15:32:23.93.

    1 |       %   TEST PROGRAM # 2   %                                                 |
    2 |     MODULE =                                                                   |
  PROD. #   4    <HEADING> ::= MODULE
    3 |         BEGIN  GLOBAL  AA<4>, AB<7>, AC ;                                      |
  PROD. #   78    <BL HEAD4> ::= BEGIN
  PROD. #   74    <BL HEAD1> ::= <BL HEAD4>
  PROD. #   220    <ALLOCATE TYPE> ::= GLOBAL
  PROD. #   218    <ALLOCATE HEAD> ::= <ALLOCATE TYPE>
  PROD. #   216    <ALLOC HEADING> ::= <ALLOCATE HEAD>
  PROD. #   64    <STRUCTURE ACCESS HEAD> ::= <IDENTIFIER><
  PROD. #   58    <LITERAL> ::= <NUMBER>
  PROD. #   47    <P1> ::= <LITERAL>
  PROD. #   45    <P3> ::= <P1>
  PROD. #   43    <P4> ::= <P3>
  PROD. #   39    <P5> ::= <P4>
  PROD. #   35    <P6> ::= <P5>
  PROD. #   33    <P7> ::= <P6>
  PROD. #   31    <P8> ::= <P7>
  PROD. #   29    <P9> ::= <P8>
  PROD. #   27    <P10> ::= <P9>
  PROD. #   24    <P11> ::= <P10>
  PROD. #   21    <SIMPLE EXPRESSION> ::= <P11>
  PROD. #   11    <BALANCED EXPRESSION> ::= <SIMPLE EXPRESSION>
  PROD. #   9    <UNLABELLED EXPR> ::= <BALANCED EXPRESSION>
  PROD. #   7    <E> ::= <UNLABELLED EXPR>
  PROD. #   63    <STRUCTURE ACCESS> ::= <STRUCTURE ACCESS HEAD><E>>
  PROD. #   210    <ALLOC1> ::= <ALLOC HEADING><STRUCTURE ACCESS>
  PROD. #   219    <ALLOCATE HEAD> ::= <ALLOC1>,
  PROD. #   216    <ALLOC HEADING> ::= <ALLOCATE HEAD>
  PROD. #   64    <STRUCTURE ACCESS HEAD> ::= <IDENTIFIER><
  PROD. #   58    <LITERAL> ::= <NUMBER>
  PROD. #   47    <P1> ::= <LITERAL>
  PROD. #   45    <P3> ::= <P1>
```

```
PROD. #  43   <P4> ::= <P3>
PROD. #  39   <P5> ::= <P4>
PROD. #  35   <P6> ::= <P5>
PROD. #  33   <P7> ::= <P6>
PROD. #  31   <P8> ::= <P7>
PROD. #  29   <P9> ::= <P8>
PROD. #  27   <P10> ::= <P9>
PROD. #  24   <P11> ::= <P10>
PROD. #  21   <SIMPLE EXPRESSION> ::= <P11>
PROD. #  11   <BALANCED EXPRESSION> ::= <SIMPLE EXPRESSION>
PROD. #  9   <UNLABELLED EXPR> ::= <BALANCED EXPRESSION>
PROD. #  7   <E> ::= <UNLABELLED EXPR>
PROD. #  63   <STRUCTURE ACCESS> ::= <STRUCTURE ACCESS HEAD><E>>
PROD. #  210   <ALLOC1> ::= <ALLOC HEADING><STRUCTURE ACCESS>
PROD. #  219   <ALLOCATE HEAD> ::= <ALLOC1>,
PROD. #  216   <ALLOC HEADING> ::= <ALLOCATE HEAD>
PROD. #  209   <ALLOC1> ::= <ALLOC HEADING><IDENTIFIER>
PROD. #  207   <ALLOCATION DECLARATION> ::= <ALLOC1>
PROD. #  177   <TYPE DECLARATION> ::= <ALLOCATION DECLARATION>
PROD. #  165   <DECLARATION> ::= <TYPE DECLARATION>
 4 |         STRUCTURE  VEC3 <I,J,K> = <I*J-K> .VEC3 + .I * J + (.J-3) + .K;         |
PROD. #  75   <BL HEAD1> ::= <BL HEAD1><DECLARATION>;
PROD. #  188   <STRUCTURE NAME> ::= STRUCTURE<IDENTIFIER>
PROD. #  189   <BRAC ID> ::= <
PROD. #  190   <BRAC ID> ::= <BRAC ID><IDENTIFIER>,
PROD. #  190   <BRAC ID> ::= <BRAC ID><IDENTIFIER>,
PROD. #  187   <STRUCTURE HEAD> ::= <STRUCTURE NAME><BRAC ID><IDENTIFIER>>
PROD. #  57   <LITERAL> ::= <IDENTIFIER>
PROD. #  47   <P1> ::= <LITERAL>
PROD. #  45   <P3> ::= <P1>
PROD. #  43   <P4> ::= <P3>
PROD. #  39   <P5> ::= <P4>
PROD. #  57   <LITERAL> ::= <IDENTIFIER>
PROD. #  47   <P1> ::= <LITERAL>
PROD. #  45   <P3> ::= <P1>
PROD. #  43   <P4> ::= <P3>
PROD. #  40   <P5> ::= <P5>*<P4>
PROD. #  35   <P6> ::= <P5>
PROD. #  57   <LITERAL> ::= <IDENTIFIER>
PROD. #  47   <P1> ::= <LITERAL>
PROD. #  45   <P3> ::= <P1>
PROD. #  43   <P4> ::= <P3>
PROD. #  39   <P5> ::= <P4>
PROD. #  38   <P6> ::= <P6>-<P5>
PROD. #  33   <P7> ::= <P6>
PROD. #  31   <P8> ::= <P7>
PROD. #  29   <P9> ::= <P8>
```

```
PROD. #  27   <P10> ::= <P9>
PROD. #  24   <P11> ::= <P10>
PROD. #  21   <SIMPLE EXPRESSION> ::= <P11>
PROD. #  11   <BALANCED EXPRESSION> ::= <SIMPLE EXPRESSION>
PROD. #  9   <UNLABELLED EXPR> ::= <BALANCED EXPRESSION>
PROD. #  7   <E> ::= <UNLABELLED EXPR>
PROD. #  120   <BRACED E> ::= <<E>>
PROD. #  57   <LITERAL> ::= <IDENTIFIER>
PROD. #  47   <P1> ::= <LITERAL>
PROD. #  45   <P3> ::= <P1>
PROD. #  46   <P3> ::= .<P3>
PROD. #  43   <P4> ::= <P3>
PROD. #  39   <P5> ::= <P4>
PROD. #  35   <P6> ::= <P5>
PROD. #  57   <LITERAL> ::= <IDENTIFIER>
PROD. #  47   <P1> ::= <LITERAL>
PROD. #  45   <P3> ::= <P1>
PROD. #  46   <P3> ::= .<P3>
PROD. #  43   <P4> ::= <P3>
PROD. #  39   <P5> ::= <P4>
PROD. #  57   <LITERAL> ::= <IDENTIFIER>
PROD. #  47   <P1> ::= <LITERAL>
PROD. #  45   <P3> ::= <P1>
PROD. #  43   <P4> ::= <P3>
PROD. #  40   <P5> ::= <P5>*<P4>
PROD. #  37   <P6> ::= <P6>+<P5>
PROD. #  79   <BL HEAD4> ::= (
PROD. #  74   <BL HEAD1> ::= <BL HEAD4>
PROD. #  76   <BL HEAD2> ::= <BL HEAD1>
PROD. #  57   <LITERAL> ::= <IDENTIFIER>
PROD. #  47   <P1> ::= <LITERAL>
PROD. #  45   <P3> ::= <P1>
PROD. #  46   <P3> ::= .<P3>
PROD. #  43   <P4> ::= <P3>
PROD. #  39   <P5> ::= <P4>
PROD. #  35   <P6> ::= <P5>
PROD. #  58   <LITERAL> ::= <NUMBER>
PROD. #  47   <P1> ::= <LITERAL>
PROD. #  45   <P3> ::= <P1>
PROD. #  43   <P4> ::= <P3>
PROD. #  39   <P5> ::= <P4>
PROD. #  38   <P6> ::= <P6>-<P5>
PROD. #  33   <P7> ::= <P6>
PROD. #  31   <P8> ::= <P7>
PROD. #  29   <P9> ::= <P8>
PROD. #  27   <P10> ::= <P9>
PROD. #  24   <P11> ::= <P10>
PROD. #  21   <SIMPLE EXPRESSION> ::= <P11>
```

```
PROD. #  11   <BALANCED EXPRESSION> ::= <SIMPLE EXPRESSION>
PROD. #  9   <UNLABELLED EXPR> ::= <BALANCED EXPRESSION>
PROD. #  7   <E> ::= <UNLABELLED EXPR>
PROD. #  73   <BLOCK HEAD> ::= <BL HEAD2><E>
PROD. #  81   <BLOCK END> ::= )
PROD. #  70   <BLOCK> ::= <BLOCK HEAD><BLOCK END>
PROD. #  50   <P1> ::= <BLOCK>
PROD. #  45   <P3> ::= <P1>
PROD. #  43   <P4> ::= <P3>
PROD. #  39   <P5> ::= <P4>
PROD. #  37   <P6> ::= <P6>+<P5>
PROD. #  57   <LITERAL> ::= <IDENTIFIER>
PROD. #  47   <P1> ::= <LITERAL>
PROD. #  45   <P3> ::= <P1>
PROD. #  46   <P3> ::= .<P3>
PROD. #  43   <P4> ::= <P3>
PROD. #  39   <P5> ::= <P4>
PROD. #  37   <P6> ::= <P6>+<P5>
PROD. #  33   <P7> ::= <P6>
PROD. #  31   <P8> ::= <P7>
PROD. #  29   <P9> ::= <P8>
PROD. #  27   <P10> ::= <P9>
PROD. #  24   <P11> ::= <P10>
PROD. #  21   <SIMPLE EXPRESSION> ::= <P11>
PROD. #  11   <BALANCED EXPRESSION> ::= <SIMPLE EXPRESSION>
PROD. #  9   <UNLABELLED EXPR> ::= <BALANCED EXPRESSION>
PROD. #  7   <E> ::= <UNLABELLED EXPR>
PROD. #  191   <STRUCTURE END> ::= =<BRACED E><E>
PROD. #  185   <STRUCTURE DECLARATION> ::= <STRUCTURE HEAD><STRUCTURE END>
PROD. #  176   <TYPE DECLARATION> ::= <STRUCTURE DECLARATION>
PROD. #  165   <DECLARATION> ::= <TYPE DECLARATION>
 5 |
 6 |          BEGIN  LOCAL  VEC3 CA<3,4,2>, CB:CC   ;
PROD. #  75   <BL HEAD1> ::= <BL HEAD1><DECLARATION>;
PROD. #  76   <BL HEAD2> ::= <BL HEAD1>
PROD. #  78   <BL HEAD4> ::= BEGIN
PROD. #  74   <BL HEAD1> ::= <BL HEAD4>
PROD. #  222   <ALLOCATE TYPE> ::= LOCAL
PROD. #  218   <ALLOCATE HEAD> ::= <ALLOCATE TYPE>
PROD. #  216   <ALLOC HEADING> ::= <ALLOCATE HEAD>
PROD. #  64   <STRUCTURE ACCESS HEAD> ::= <IDENTIFIER><
PROD. #  58   <LITERAL> ::= <NUMBER>
PROD. #  47   <P1> ::= <LITERAL>
PROD. #  45   <P3> ::= <P1>
PROD. #  43   <P4> ::= <P3>
PROD. #  39   <P5> ::= <P4>
PROD. #  35   <P6> ::= <P5>
PROD. #  33   <P7> ::= <P6>
```

```
PROD. #  31   <P8> ::= <P7>
PROD. #  29   <P9> ::= <P8>
PROD. #  27   <P10> ::= <P9>
PROD. #  24   <P11> ::= <P10>
PROD. #  21   <SIMPLE EXPRESSION> ::= <P11>
PROD. #  11   <BALANCED EXPRESSION> ::= <SIMPLE EXPRESSION>
PROD. #  9   <UNLABELLED EXPR> ::= <BALANCED EXPRESSION>
PROD. #  7   <E> ::= <UNLABELLED EXPR>
PROD. #  65   <STRUCTURE ACCESS HEAD> ::= <STRUCTURE ACCESS HEAD><E>,
PROD. #  58   <LITERAL> ::= <NUMBER>
PROD. #  47   <P1> ::= <LITERAL>
PROD. #  45   <P3> ::= <P1>
PROD. #  43   <P4> ::= <P3>
PROD. #  39   <P5> ::= <P4>
PROD. #  35   <P6> ::= <P5>
PROD. #  33   <P7> ::= <P6>
PROD. #  31   <P8> ::= <P7>
PROD. #  29   <P9> ::= <P8>
PROD. #  27   <P10> ::= <P9>
PROD. #  24   <P11> ::= <P10>
PROD. #  21   <SIMPLE EXPRESSION> ::= <P11>
PROD. #  11   <BALANCED EXPRESSION> ::= <SIMPLE EXPRESSION>
PROD. #  9   <UNLABELLED EXPR> ::= <BALANCED EXPRESSION>
PROD. #  7   <E> ::= <UNLABELLED EXPR>
PROD. #  65   <STRUCTURE ACCESS HEAD> ::= <STRUCTURE ACCESS HEAD><E>,
PROD. #  58   <LITERAL> ::= <NUMBER>
PROD. #  47   <P1> ::= <LITERAL>
PROD. #  45   <P3> ::= <P1>
PROD. #  43   <P4> ::= <P3>
PROD. #  39   <P5> ::= <P4>
PROD. #  35   <P6> ::= <P5>
PROD. #  33   <P7> ::= <P6>
PROD. #  31   <P8> ::= <P7>
PROD. #  29   <P9> ::= <P8>
PROD. #  27   <P10> ::= <P9>
PROD. #  24   <P11> ::= <P10>
PROD. #  21   <SIMPLE EXPRESSION> ::= <P11>
PROD. #  11   <BALANCED EXPRESSION> ::= <SIMPLE EXPRESSION>
PROD. #  9   <UNLABELLED EXPR> ::= <BALANCED EXPRESSION>
PROD. #  7   <E> ::= <UNLABELLED EXPR>
PROD. #  63   <STRUCTURE ACCESS> ::= <STRUCTURE ACCESS HEAD><E>>
PROD. #  215   <STRUCTURE ALLOC> ::= <IDENTIFIER><STRUCTURE ACCESS>
PROD. #  208   <ALLOC1> ::= <ALLOC HEADING><STRUCTURE ALLOC>
PROD. #  219   <ALLOCATE HEAD> ::= <ALLOC1>,
PROD. #  216   <ALLOC HEADING> ::= <ALLOCATE HEAD>
PROD. #  209   <ALLOC1> ::= <ALLOC HEADING><IDENTIFIER>
PROD. #  213   <ALLOC0> ::= <ALLOC1>:
PROD. #  211   <ALLOC1> ::= <ALLOC0><IDENTIFIER>
```

```
PROD. #  207   <ALLOCATION DECLARATION> ::= <ALLOC1>
PROD. #  177   <TYPE DECLARATION> ::= <ALLOCATION DECLARATION>
PROD. #  165   <DECLARATION> ::= <TYPE DECLARATION>
 7 |          ROUTINE CATZ =  ( LOCAL  DA:DB:DC;    ?  A NONSENSE FUNCTION          |
PROD. #  75   <BL HEAD1> ::= <BL HEAD1><DECLARATION>;
PROD. #  201   <FUNCTION TYPE> ::= ROUTINE
PROD. #  199   <PARAMLESS FUNCTION> ::= <FUNCTION TYPE><IDENTIFIER>
PROD. #  195   <TYPE FUNCTION HEAD> ::= <PARAMLESS FUNCTION>
PROD. #  79   <BL HEAD4> ::= (
PROD. #  74   <BL HEAD1> ::= <BL HEAD4>
PROD. #  222   <ALLOCATE TYPE> ::= LOCAL
PROD. #  218   <ALLOCATE HEAD> ::= <ALLOCATE TYPE>
PROD. #  216   <ALLOC HEADING> ::= <ALLOCATE HEAD>
PROD. #  209   <ALLOC1> ::= <ALLOC HEADING><IDENTIFIER>
PROD. #  213   <ALLOC0> ::= <ALLOC1>:
PROD. #  211   <ALLOC1> ::= <ALLOC0><IDENTIFIER>
PROD. #  213   <ALLOC0> ::= <ALLOC1>:
PROD. #  211   <ALLOC1> ::= <ALLOC0><IDENTIFIER>
PROD. #  207   <ALLOCATION DECLARATION> ::= <ALLOC1>
PROD. #  177   <TYPE DECLARATION> ::= <ALLOCATION DECLARATION>
PROD. #  165   <DECLARATION> ::= <TYPE DECLARATION>
 8 |           DA:=.DB:=..DA;     .DC MOD .DB ) ;                                   |
PROD. #  75   <BL HEAD1> ::= <BL HEAD1><DECLARATION>;
PROD. #  76   <BL HEAD2> ::= <BL HEAD1>
PROD. #  57   <LITERAL> ::= <IDENTIFIER>
PROD. #  47   <P1> ::= <LITERAL>
PROD. #  45   <P3> ::= <P1>
PROD. #  43   <P4> ::= <P3>
PROD. #  39   <P5> ::= <P4>
PROD. #  35   <P6> ::= <P5>
PROD. #  33   <P7> ::= <P6>
PROD. #  31   <P8> ::= <P7>
PROD. #  29   <P9> ::= <P8>
PROD. #  27   <P10> ::= <P9>
PROD. #  24   <P11> ::= <P10>
PROD. #  57   <LITERAL> ::= <IDENTIFIER>
PROD. #  47   <P1> ::= <LITERAL>
PROD. #  45   <P3> ::= <P1>
PROD. #  46   <P3> ::= .<P3>
PROD. #  43   <P4> ::= <P3>
PROD. #  39   <P5> ::= <P4>
PROD. #  35   <P6> ::= <P5>
PROD. #  33   <P7> ::= <P6>
PROD. #  31   <P8> ::= <P7>
PROD. #  29   <P9> ::= <P8>
PROD. #  27   <P10> ::= <P9>
PROD. #  24   <P11> ::= <P10>
PROD. #  57   <LITERAL> ::= <IDENTIFIER>
```

```
PROD. #  47   <P1> ::= <LITERAL>
PROD. #  45   <P3> ::= <P1>
PROD. #  46   <P3> ::= .<P3>
PROD. #  46   <P3> ::= .<P3>
PROD. #  43   <P4> ::= <P3>
PROD. #  39   <P5> ::= <P4>
PROD. #  35   <P6> ::= <P5>
PROD. #  33   <P7> ::= <P6>
PROD. #  31   <P8> ::= <P7>
PROD. #  29   <P9> ::= <P8>
PROD. #  27   <P10> ::= <P9>
PROD. #  24   <P11> ::= <P10>
PROD. #  21   <SIMPLE EXPRESSION> ::= <P11>
PROD. #  11   <BALANCED EXPRESSION> ::= <SIMPLE EXPRESSION>
PROD. #  9   <UNLABELLED EXPR> ::= <BALANCED EXPRESSION>
PROD. #  7   <E> ::= <UNLABELLED EXPR>
PROD. #  23   <RIGHT PART> ::= :=<E>
PROD. #  22   <SIMPLE EXPRESSION> ::= <P11><RIGHT PART>
PROD. #  11   <BALANCED EXPRESSION> ::= <SIMPLE EXPRESSION>
PROD. #  9   <UNLABELLED EXPR> ::= <BALANCED EXPRESSION>
PROD. #  7   <E> ::= <UNLABELLED EXPR>
PROD. #  23   <RIGHT PART> ::= :=<E>
PROD. #  22   <SIMPLE EXPRESSION> ::= <P11><RIGHT PART>
PROD. #  11   <BALANCED EXPRESSION> ::= <SIMPLE EXPRESSION>
PROD. #  9   <UNLABELLED EXPR> ::= <BALANCED EXPRESSION>
PROD. #  7   <E> ::= <UNLABELLED EXPR>
PROD. #  77   <BL HEAD2> ::= <BL HEAD2><E>;
PROD. #  57   <LITERAL> ::= <IDENTIFIER>
PROD. #  47   <P1> ::= <LITERAL>
PROD. #  45   <P3> ::= <P1>
PROD. #  46   <P3> ::= .<P3>
PROD. #  43   <P4> ::= <P3>
PROD. #  39   <P5> ::= <P4>
PROD. #  57   <LITERAL> ::= <IDENTIFIER>
PROD. #  47   <P1> ::= <LITERAL>
PROD. #  45   <P3> ::= <P1>
PROD. #  46   <P3> ::= .<P3>
PROD. #  43   <P4> ::= <P3>
PROD. #  42   <P5> ::= <P5>MOD<P4>
PROD. #  35   <P6> ::= <P5>
PROD. #  33   <P7> ::= <P6>
PROD. #  31   <P8> ::= <P7>
PROD. #  29   <P9> ::= <P8>
PROD. #  27   <P10> ::= <P9>
PROD. #  24   <P11> ::= <P10>
PROD. #  21   <SIMPLE EXPRESSION> ::= <P11>
PROD. #  11   <BALANCED EXPRESSION> ::= <SIMPLE EXPRESSION>
PROD. #  9   <UNLABELLED EXPR> ::= <BALANCED EXPRESSION>
```

```
PROD. #  7   <E> ::= <UNLABELLED EXPR>
PROD. #  73   <BLOCK HEAD> ::= <BL HEAD2><E>
PROD. #  81   <BLOCK END> ::= )
PROD. #  70   <BLOCK> ::= <BLOCK HEAD><BLOCK END>
PROD. #  50   <P1> ::= <BLOCK>
PROD. #  45   <P3> ::= <P1>
PROD. #  43   <P4> ::= <P3>
PROD. #  39   <P5> ::= <P4>
PROD. #  35   <P6> ::= <P5>
PROD. #  33   <P7> ::= <P6>
PROD. #  31   <P8> ::= <P7>
PROD. #  29   <P9> ::= <P8>
PROD. #  27   <P10> ::= <P9>
PROD. #  24   <P11> ::= <P10>
PROD. #  21   <SIMPLE EXPRESSION> ::= <P11>
PROD. #  11   <BALANCED EXPRESSION> ::= <SIMPLE EXPRESSION>
PROD. #  9   <UNLABELLED EXPR> ::= <BALANCED EXPRESSION>
PROD. #  7   <E> ::= <UNLABELLED EXPR>
PROD. #  174   <EQE> ::= =<E>
PROD. #  194   <TYPE FUNCTION> ::= <TYPE FUNCTION HEAD><EQE>
PROD. #  192   <FUNCTION DECLARATION> ::= <TYPE FUNCTION>
PROD. #  175   <TYPE DECLARATION> ::= <FUNCTION DECLARATION>
PROD. #  165   <DECLARATION> ::= <TYPE DECLARATION>
 9 |          BIND Y = .CA<.CB, .CC, 2>;
PROD. #  75   <BL HEAD1> ::= <BL HEAD1><DECLARATION>;
PROD. #  229   <BIND HEADING> ::= BIND
PROD. #  227   <BIND HEAD> ::= <BIND HEADING>
PROD. #  217   <ALLOC HEADING> ::= <BIND HEAD>
PROD. #  209   <ALLOC1> ::= <ALLOC HEADING><IDENTIFIER>
PROD. #  64   <STRUCTURE ACCESS HEAD> ::= <IDENTIFIER><
PROD. #  57   <LITERAL> ::= <IDENTIFIER>
PROD. #  47   <P1> ::= <LITERAL>
PROD. #  45   <P3> ::= <P1>
PROD. #  46   <P3> ::= .<P3>
PROD. #  43   <P4> ::= <P3>
PROD. #  39   <P5> ::= <P4>
PROD. #  35   <P6> ::= <P5>
PROD. #  33   <P7> ::= <P6>
PROD. #  31   <P8> ::= <P7>
PROD. #  29   <P9> ::= <P8>
PROD. #  27   <P10> ::= <P9>
PROD. #  24   <P11> ::= <P10>
PROD. #  21   <SIMPLE EXPRESSION> ::= <P11>
PROD. #  11   <BALANCED EXPRESSION> ::= <SIMPLE EXPRESSION>
PROD. #  9   <UNLABELLED EXPR> ::= <BALANCED EXPRESSION>
PROD. #  7   <E> ::= <UNLABELLED EXPR>
PROD. #  65   <STRUCTURE ACCESS HEAD> ::= <STRUCTURE ACCESS HEAD><E>,
PROD. #  57   <LITERAL> ::= <IDENTIFIER>
```

```
PROD. #  47   <P1> ::= <LITERAL>
PROD. #  45   <P3> ::= <P1>
PROD. #  46   <P3> ::= .<P3>
PROD. #  43   <P4> ::= <P3>
PROD. #  39   <P5> ::= <P4>
PROD. #  35   <P6> ::= <P5>
PROD. #  33   <P7> ::= <P6>
PROD. #  31   <P8> ::= <P7>
PROD. #  29   <P9> ::= <P8>
PROD. #  27   <P10> ::= <P9>
PROD. #  24   <P11> ::= <P10>
PROD. #  21   <SIMPLE EXPRESSION> ::= <P11>
PROD. #  11   <BALANCED EXPRESSION> ::= <SIMPLE EXPRESSION>
PROD. #  9   <UNLABELLED EXPR> ::= <BALANCED EXPRESSION>
PROD. #  7   <E> ::= <UNLABELLED EXPR>
PROD. #  65   <STRUCTURE ACCESS HEAD> ::= <STRUCTURE ACCESS HEAD><E>,
PROD. #  58   <LITERAL> ::= <NUMBER>
PROD. #  47   <P1> ::= <LITERAL>
PROD. #  45   <P3> ::= <P1>
PROD. #  43   <P4> ::= <P3>
PROD. #  39   <P5> ::= <P4>
PROD. #  35   <P6> ::= <P5>
PROD. #  33   <P7> ::= <P6>
PROD. #  31   <P8> ::= <P7>
PROD. #  29   <P9> ::= <P8>
PROD. #  27   <P10> ::= <P9>
PROD. #  24   <P11> ::= <P10>
PROD. #  21   <SIMPLE EXPRESSION> ::= <P11>
PROD. #  11   <BALANCED EXPRESSION> ::= <SIMPLE EXPRESSION>
PROD. #  9   <UNLABELLED EXPR> ::= <BALANCED EXPRESSION>
PROD. #  7   <E> ::= <UNLABELLED EXPR>
PROD. #  63   <STRUCTURE ACCESS> ::= <STRUCTURE ACCESS HEAD><E>>
PROD. #  48   <P1> ::= <STRUCTURE ACCESS>
PROD. #  45   <P3> ::= <P1>
PROD. #  46   <P3> ::= .<P3>
PROD. #  43   <P4> ::= <P3>
PROD. #  39   <P5> ::= <P4>
PROD. #  35   <P6> ::= <P5>
PROD. #  33   <P7> ::= <P6>
PROD. #  31   <P8> ::= <P7>
PROD. #  29   <P9> ::= <P8>
PROD. #  27   <P10> ::= <P9>
PROD. #  24   <P11> ::= <P10>
PROD. #  21   <SIMPLE EXPRESSION> ::= <P11>
PROD. #  11   <BALANCED EXPRESSION> ::= <SIMPLE EXPRESSION>
PROD. #  9   <UNLABELLED EXPR> ::= <BALANCED EXPRESSION>
PROD. #  7   <E> ::= <UNLABELLED EXPR>
PROD. #  174   <EQE> ::= =<E>
```

```
PROD. #  231   <ALLOC2> ::= <ALLOC1><EQE>
PROD. #  230   <BIND DECLARATION> ::= <ALLOC2>
PROD. #  178   <TYPE DECLARATION> ::= <BIND DECLARATION>
PROD. #  165   <DECLARATION> ::= <TYPE DECLARATION>
10 |           CA<1,2,3> :=  .CATZ() ;
PROD. #  75   <BL HEAD1> ::= <BL HEAD1><DECLARATION>;
PROD. #  76   <BL HEAD2> ::= <BL HEAD1>
PROD. #  64   <STRUCTURE ACCESS HEAD> ::= <IDENTIFIER><
PROD. #  58   <LITERAL> ::= <NUMBER>
PROD. #  47   <P1> ::= <LITERAL>
PROD. #  45   <P3> ::= <P1>
PROD. #  43   <P4> ::= <P3>
PROD. #  39   <P5> ::= <P4>
PROD. #  35   <P6> ::= <P5>
PROD. #  33   <P7> ::= <P6>
PROD. #  31   <P8> ::= <P7>
PROD. #  29   <P9> ::= <P8>
PROD. #  27   <P10> ::= <P9>
PROD. #  24   <P11> ::= <P10>
PROD. #  21   <SIMPLE EXPRESSION> ::= <P11>
PROD. #  11   <BALANCED EXPRESSION> ::= <SIMPLE EXPRESSION>
PROD. #  9   <UNLABELLED EXPR> ::= <BALANCED EXPRESSION>
PROD. #  7   <E> ::= <UNLABELLED EXPR>
PROD. #  65   <STRUCTURE ACCESS HEAD> ::= <STRUCTURE ACCESS HEAD><E>,
PROD. #  58   <LITERAL> ::= <NUMBER>
PROD. #  47   <P1> ::= <LITERAL>
PROD. #  45   <P3> ::= <P1>
PROD. #  43   <P4> ::= <P3>
PROD. #  39   <P5> ::= <P4>
PROD. #  35   <P6> ::= <P5>
PROD. #  33   <P7> ::= <P6>
PROD. #  31   <P8> ::= <P7>
PROD. #  29   <P9> ::= <P8>
PROD. #  27   <P10> ::= <P9>
PROD. #  24   <P11> ::= <P10>
PROD. #  21   <SIMPLE EXPRESSION> ::= <P11>
PROD. #  11   <BALANCED EXPRESSION> ::= <SIMPLE EXPRESSION>
PROD. #  9   <UNLABELLED EXPR> ::= <BALANCED EXPRESSION>
PROD. #  7   <E> ::= <UNLABELLED EXPR>
PROD. #  65   <STRUCTURE ACCESS HEAD> ::= <STRUCTURE ACCESS HEAD><E>,
PROD. #  58   <LITERAL> ::= <NUMBER>
PROD. #  47   <P1> ::= <LITERAL>
PROD. #  45   <P3> ::= <P1>
PROD. #  43   <P4> ::= <P3>
PROD. #  39   <P5> ::= <P4>
PROD. #  35   <P6> ::= <P5>
PROD. #  33   <P7> ::= <P6>
PROD. #  31   <P8> ::= <P7>
```

```
PROD. #  29   <P9> ::= <P8>
PROD. #  27   <P10> ::= <P9>
PROD. #  24   <P11> ::= <P10>
PROD. #  21   <SIMPLE EXPRESSION> ::= <P11>
PROD. #  11   <BALANCED EXPRESSION> ::= <SIMPLE EXPRESSION>
PROD. #  9   <UNLABELLED EXPR> ::= <BALANCED EXPRESSION>
PROD. #  7   <E> ::= <UNLABELLED EXPR>
PROD. #  63   <STRUCTURE ACCESS> ::= <STRUCTURE ACCESS HEAD><E>>
PROD. #  48   <P1> ::= <STRUCTURE ACCESS>
PROD. #  45   <P3> ::= <P1>
PROD. #  43   <P4> ::= <P3>
PROD. #  39   <P5> ::= <P4>
PROD. #  35   <P6> ::= <P5>
PROD. #  33   <P7> ::= <P6>
PROD. #  31   <P8> ::= <P7>
PROD. #  29   <P9> ::= <P8>
PROD. #  27   <P10> ::= <P9>
PROD. #  24   <P11> ::= <P10>
PROD. #  68   <FUNCTION HEAD> ::= <IDENTIFIER>(
PROD. #  67   <FUNCTION CALL> ::= <FUNCTION HEAD>)
PROD. #  49   <P1> ::= <FUNCTION CALL>
PROD. #  45   <P3> ::= <P1>
PROD. #  46   <P3> ::= .<P3>
PROD. #  43   <P4> ::= <P3>
PROD. #  39   <P5> ::= <P4>
PROD. #  35   <P6> ::= <P5>
PROD. #  33   <P7> ::= <P6>
PROD. #  31   <P8> ::= <P7>
PROD. #  29   <P9> ::= <P8>
PROD. #  27   <P10> ::= <P9>
PROD. #  24   <P11> ::= <P10>
PROD. #  21   <SIMPLE EXPRESSION> ::= <P11>
PROD. #  11   <BALANCED EXPRESSION> ::= <SIMPLE EXPRESSION>
PROD. #  9   <UNLABELLED EXPR> ::= <BALANCED EXPRESSION>
PROD. #  7   <E> ::= <UNLABELLED EXPR>
PROD. #  23   <RIGHT PART> ::= :=<E>
PROD. #  22   <SIMPLE EXPRESSION> ::= <P11><RIGHT PART>
PROD. #  11   <BALANCED EXPRESSION> ::= <SIMPLE EXPRESSION>
PROD. #  9   <UNLABELLED EXPR> ::= <BALANCED EXPRESSION>
PROD. #  7   <E> ::= <UNLABELLED EXPR>
11 |            AA<.CB> := .CB XOR .CC
PROD. #  77   <BL HEAD2> ::= <BL HEAD2><E>;
PROD. #  64   <STRUCTURE ACCESS HEAD> ::= <IDENTIFIER><
PROD. #  57   <LITERAL> ::= <IDENTIFIER>
PROD. #  47   <P1> ::= <LITERAL>
PROD. #  45   <P3> ::= <P1>
PROD. #  46   <P3> ::= .<P3>
PROD. #  43   <P4> ::= <P3>
```

```
PROD. #  39   <P5> ::= <P4>
PROD. #  35   <P6> ::= <P5>
PROD. #  33   <P7> ::= <P6>
PROD. #  31   <P8> ::= <P7>
PROD. #  29   <P9> ::= <P8>
PROD. #  27   <P10> ::= <P9>
PROD. #  24   <P11> ::= <P10>
PROD. #  21   <SIMPLE EXPRESSION> ::= <P11>
PROD. #  11   <BALANCED EXPRESSION> ::= <SIMPLE EXPRESSION>
PROD. #  9   <UNLABELLED EXPR> ::= <BALANCED EXPRESSION>
PROD. #  7   <E> ::= <UNLABELLED EXPR>
PROD. #  63   <STRUCTURE ACCESS> ::= <STRUCTURE ACCESS HEAD><E>>
PROD. #  48   <P1> ::= <STRUCTURE ACCESS>
PROD. #  45   <P3> ::= <P1>
PROD. #  43   <P4> ::= <P3>
PROD. #  39   <P5> ::= <P4>
PROD. #  35   <P6> ::= <P5>
PROD. #  33   <P7> ::= <P6>
PROD. #  31   <P8> ::= <P7>
PROD. #  29   <P9> ::= <P8>
PROD. #  27   <P10> ::= <P9>
PROD. #  24   <P11> ::= <P10>
PROD. #  57   <LITERAL> ::= <IDENTIFIER>
PROD. #  47   <P1> ::= <LITERAL>
PROD. #  45   <P3> ::= <P1>
PROD. #  46   <P3> ::= .<P3>
PROD. #  43   <P4> ::= <P3>
PROD. #  39   <P5> ::= <P4>
PROD. #  35   <P6> ::= <P5>
PROD. #  33   <P7> ::= <P6>
PROD. #  31   <P8> ::= <P7>
PROD. #  29   <P9> ::= <P8>
PROD. #  27   <P10> ::= <P9>
PROD. #  24   <P11> ::= <P10>
12 |         END;
PROD. #  57   <LITERAL> ::= <IDENTIFIER>
PROD. #  47   <P1> ::= <LITERAL>
PROD. #  45   <P3> ::= <P1>
PROD. #  46   <P3> ::= .<P3>
PROD. #  43   <P4> ::= <P3>
PROD. #  39   <P5> ::= <P4>
PROD. #  35   <P6> ::= <P5>
PROD. #  33   <P7> ::= <P6>
PROD. #  31   <P8> ::= <P7>
PROD. #  29   <P9> ::= <P8>
PROD. #  27   <P10> ::= <P9>
PROD. #  25   <P11> ::= <P11>XOR<P10>
PROD. #  21   <SIMPLE EXPRESSION> ::= <P11>
```

```
PROD. #  11   <BALANCED EXPRESSION> ::= <SIMPLE EXPRESSION>
PROD. #  9   <UNLABELLED EXPR> ::= <BALANCED EXPRESSION>
PROD. #  7   <E> ::= <UNLABELLED EXPR>
PROD. #  23   <RIGHT PART> ::= :=<E>
PROD. #  22   <SIMPLE EXPRESSION> ::= <P11><RIGHT PART>
PROD. #  11   <BALANCED EXPRESSION> ::= <SIMPLE EXPRESSION>
PROD. #  9   <UNLABELLED EXPR> ::= <BALANCED EXPRESSION>
PROD. #  7   <E> ::= <UNLABELLED EXPR>
PROD. #  73   <BLOCK HEAD> ::= <BL HEAD2><E>
PROD. #  80   <BLOCK END> ::= END
PROD. #  70   <BLOCK> ::= <BLOCK HEAD><BLOCK END>
PROD. #  50   <P1> ::= <BLOCK>
PROD. #  45   <P3> ::= <P1>
PROD. #  43   <P4> ::= <P3>
PROD. #  39   <P5> ::= <P4>
PROD. #  35   <P6> ::= <P5>
PROD. #  33   <P7> ::= <P6>
PROD. #  31   <P8> ::= <P7>
PROD. #  29   <P9> ::= <P8>
PROD. #  27   <P10> ::= <P9>
PROD. #  24   <P11> ::= <P10>
PROD. #  21   <SIMPLE EXPRESSION> ::= <P11>
PROD. #  11   <BALANCED EXPRESSION> ::= <SIMPLE EXPRESSION>
PROD. #  9   <UNLABELLED EXPR> ::= <BALANCED EXPRESSION>
PROD. #  7   <E> ::= <UNLABELLED EXPR>
13 |        AA<3> := IF NOT AC THEN AC|(-2) ELSE AC|3
PROD. #  77   <BL HEAD2> ::= <BL HEAD2><E>;
PROD. #  64   <STRUCTURE ACCESS HEAD> ::= <IDENTIFIER><
PROD. #  58   <LITERAL> ::= <NUMBER>
PROD. #  47   <P1> ::= <LITERAL>
PROD. #  45   <P3> ::= <P1>
PROD. #  43   <P4> ::= <P3>
PROD. #  39   <P5> ::= <P4>
PROD. #  35   <P6> ::= <P5>
PROD. #  33   <P7> ::= <P6>
PROD. #  31   <P8> ::= <P7>
PROD. #  29   <P9> ::= <P8>
PROD. #  27   <P10> ::= <P9>
PROD. #  24   <P11> ::= <P10>
PROD. #  21   <SIMPLE EXPRESSION> ::= <P11>
PROD. #  11   <BALANCED EXPRESSION> ::= <SIMPLE EXPRESSION>
PROD. #  9   <UNLABELLED EXPR> ::= <BALANCED EXPRESSION>
PROD. #  7   <E> ::= <UNLABELLED EXPR>
PROD. #  63   <STRUCTURE ACCESS> ::= <STRUCTURE ACCESS HEAD><E>>
PROD. #  48   <P1> ::= <STRUCTURE ACCESS>
PROD. #  45   <P3> ::= <P1>
PROD. #  43   <P4> ::= <P3>
PROD. #  39   <P5> ::= <P4>
```

```
PROD. #  35   <P6> ::= <P5>
PROD. #  33   <P7> ::= <P6>
PROD. #  31   <P8> ::= <P7>
PROD. #  29   <P9> ::= <P8>
PROD. #  27   <P10> ::= <P9>
PROD. #  24   <P11> ::= <P10>
PROD. #  57   <LITERAL> ::= <IDENTIFIER>
PROD. #  47   <P1> ::= <LITERAL>
PROD. #  45   <P3> ::= <P1>
PROD. #  43   <P4> ::= <P3>
PROD. #  39   <P5> ::= <P4>
PROD. #  35   <P6> ::= <P5>
PROD. #  33   <P7> ::= <P6>
PROD. #  32   <P8> ::= NOT<P7>
PROD. #  29   <P9> ::= <P8>
PROD. #  27   <P10> ::= <P9>
PROD. #  24   <P11> ::= <P10>
PROD. #  21   <SIMPLE EXPRESSION> ::= <P11>
PROD. #  11   <BALANCED EXPRESSION> ::= <SIMPLE EXPRESSION>
PROD. #  9   <UNLABELLED EXPR> ::= <BALANCED EXPRESSION>
PROD. #  7   <E> ::= <UNLABELLED EXPR>
PROD. #  16   <IF CLAUSE> ::= IF<E>
PROD. #  159   <THEN> ::= THEN
PROD. #  57   <LITERAL> ::= <IDENTIFIER>
PROD. #  47   <P1> ::= <LITERAL>
PROD. #  45   <P3> ::= <P1>
PROD. #  43   <P4> ::= <P3>
PROD. #  79   <BL HEAD4> ::= (
PROD. #  74   <BL HEAD1> ::= <BL HEAD4>
PROD. #  76   <BL HEAD2> ::= <BL HEAD1>
PROD. #  58   <LITERAL> ::= <NUMBER>
PROD. #  47   <P1> ::= <LITERAL>
PROD. #  45   <P3> ::= <P1>
PROD. #  43   <P4> ::= <P3>
PROD. #  39   <P5> ::= <P4>
PROD. #  36   <P6> ::= -<P5>
PROD. #  33   <P7> ::= <P6>
PROD. #  31   <P8> ::= <P7>
PROD. #  29   <P9> ::= <P8>
PROD. #  27   <P10> ::= <P9>
PROD. #  24   <P11> ::= <P10>
PROD. #  21   <SIMPLE EXPRESSION> ::= <P11>
PROD. #  11   <BALANCED EXPRESSION> ::= <SIMPLE EXPRESSION>
PROD. #  9   <UNLABELLED EXPR> ::= <BALANCED EXPRESSION>
PROD. #  7   <E> ::= <UNLABELLED EXPR>
PROD. #  73   <BLOCK HEAD> ::= <BL HEAD2><E>
PROD. #  81   <BLOCK END> ::= )
PROD. #  70   <BLOCK> ::= <BLOCK HEAD><BLOCK END>
```

```
PROD. #  50   <P1> ::= <BLOCK>
PROD. #  45   <P3> ::= <P1>
PROD. #  44   <P4> ::= <P4>|<P3>
PROD. #  39   <P5> ::= <P4>
PROD. #  35   <P6> ::= <P5>
PROD. #  33   <P7> ::= <P6>
PROD. #  31   <P8> ::= <P7>
PROD. #  29   <P9> ::= <P8>
PROD. #  27   <P10> ::= <P9>
PROD. #  24   <P11> ::= <P10>
PROD. #  21   <SIMPLE EXPRESSION> ::= <P11>
PROD. #  11   <BALANCED EXPRESSION> ::= <SIMPLE EXPRESSION>
PROD. #  18   <THEN CLAUSE> ::= <THEN><BALANCED EXPRESSION>
PROD. #  17   <TEST TRUE PART> ::= <IF CLAUSE><THEN CLAUSE>
PROD. #  57   <LITERAL> ::= <IDENTIFIER>
PROD. #  47   <P1> ::= <LITERAL>
PROD. #  45   <P3> ::= <P1>
PROD. #  43   <P4> ::= <P3>
14 |     END
PROD. #  58   <LITERAL> ::= <NUMBER>
PROD. #  47   <P1> ::= <LITERAL>
PROD. #  45   <P3> ::= <P1>
PROD. #  44   <P4> ::= <P4>|<P3>
PROD. #  39   <P5> ::= <P4>
PROD. #  35   <P6> ::= <P5>
PROD. #  33   <P7> ::= <P6>
PROD. #  31   <P8> ::= <P7>
PROD. #  29   <P9> ::= <P8>
PROD. #  27   <P10> ::= <P9>
PROD. #  24   <P11> ::= <P10>
PROD. #  21   <SIMPLE EXPRESSION> ::= <P11>
PROD. #  11   <BALANCED EXPRESSION> ::= <SIMPLE EXPRESSION>
PROD. #  19   <BAL PART> ::= ELSE<BALANCED EXPRESSION>
PROD. #  13   <BALANCED EXPRESSION> ::= <TEST TRUE PART><BAL PART>
PROD. #  9   <UNLABELLED EXPR> ::= <BALANCED EXPRESSION>
PROD. #  7   <E> ::= <UNLABELLED EXPR>
PROD. #  23   <RIGHT PART> ::= :=<E>
PROD. #  22   <SIMPLE EXPRESSION> ::= <P11><RIGHT PART>
PROD. #  11   <BALANCED EXPRESSION> ::= <SIMPLE EXPRESSION>
PROD. #  9   <UNLABELLED EXPR> ::= <BALANCED EXPRESSION>
PROD. #  7   <E> ::= <UNLABELLED EXPR>
PROD. #  73   <BLOCK HEAD> ::= <BL HEAD2><E>
15 |   ELUDOM     EOF
PROD. #  80   <BLOCK END> ::= END
PROD. #  70   <BLOCK> ::= <BLOCK HEAD><BLOCK END>
PROD. #  50   <P1> ::= <BLOCK>
PROD. #  45   <P3> ::= <P1>
PROD. #  43   <P4> ::= <P3>
```

```
  PROD. #  39   <P5> ::= <P4>
  PROD. #  35   <P6> ::= <P5>
  PROD. #  33   <P7> ::= <P6>
  PROD. #  31   <P8> ::= <P7>
  PROD. #  29   <P9> ::= <P8>
  PROD. #  27   <P10> ::= <P9>
  PROD. #  24   <P11> ::= <P10>
  PROD. #  21   <SIMPLE EXPRESSION> ::= <P11>
  PROD. #  11   <BALANCED EXPRESSION> ::= <SIMPLE EXPRESSION>
  PROD. #  9   <UNLABELLED EXPR> ::= <BALANCED EXPRESSION>
  PROD. #  7   <E> ::= <UNLABELLED EXPR>
  PROD. #  174   <EQE> ::= =<E>
  PROD. #  2   <MODULE HEAD> ::= <HEADING><EQE>
  PROD. #  5   <MODULE END> ::= ELUDOM
  PROD. #  1   <MODULE> ::= <MODULE HEAD><MODULE END>
END OF CHECKING JUNE 14, 1972.  CLOCK TIME = 15:32:55.89.

15 CARDS WERE CHECKED.
NO ERRORS WERE DETECTED.


TOTAL TIME IN CHECKER      0:0:32.53.
SET UP TIME                0:0:0.39.
ACTUAL CHECKING TIME       0:0:31.68.
CLEAN-UP TIME AT END       0:0:0.46.
CHECKING RATE: 28 CARDS PER MINUTE.
```

APPENDIX C

BLISS-360 SKELETON MODIFICATIONS

```
/*   THE  BONES  OF  A  BLISS  SKELETON     */
 /*  TABLES PUNCHED BY SYNTAX ANALYZER   */
 DECLARE NSY LITERALLY '216', NT LITERALLY '83';
 DECLARE V(NSY) CHARACTER INITIAL ( '<ERROR: TOKEN = 0>', ':', '-', '+', '*',
                  .                             .
                  .                             .
                  .                             .
/* END OF CARDS PUNCHED BY SYNTAX ANALYZER */
/*DECLARATIONS FORMING THE CONSTANT DEFINITION TABLE  */
      - -  SEE SEC. IV.B.1  - -
  /*  DEFINITIONS FORMING THE DATA DICTIONARY (DICT)   */
      - -  SEE SEC. IV.B.2  - -
   /*  DEFINITIONS FOR THE INTERMEDIATE LANGUAGE TABLE IL   */
      - -  SEE SEC. IV.C.2  - -

    DECLARE LEFT LITERALLY 'MP', /* LEFT END OF PRODUCTION */
       RIGHT LITERALLY 'SP';   /* RIGHT END OF PRODUCTION */
DECLARE ALLOCIND BIT(1);  /* RESOLVE STRUCTURE ACCESS CONFLICT, =0 NOT IN,
         =1 IN ALLOCATION MODE  */
    DECLARE TESTIT BIT(8) INITIAL (0);   /*  CONTROLS THE DEBUG STATEMENTS */

 /*  DECLARATIONS FOR THE SCANNER                                         */
 /* TOKEN IS THE INDEX INTO THE VOCABULARY V() OF THE LAST SYMBOL SCANNED,
    CP IS THE POINTER TO THE LAST CHARACTER SCANNED IN THE CARDIMAGE,
    BCD IS THE LAST SYMBOL SCANNED (LITERAL CHARACTER STRING). */
 DECLARE (TOKEN, CP) FIXED, BCD CHARACTER;

 /* SET UP SOME CONVENIENT ABBREVIATIONS FOR PRINTER CONTROL */
 DECLARE EJECT_PAGE LITERALLY 'OUTPUT(1) = PAGE',
```

```
   PAGE CHARACTER INITIAL ('1'), DOUBLE CHARACTER INITIAL ('0'),
   DOUBLE_SPACE LITERALLY 'OUTPUT(1) = DOUBLE',
   X70 CHARACTER INITIAL ('
                ');

/* LENGTH OF LONGEST SYMBOL IN V */
DECLARE (RESERVED_LIMIT, MARGIN_CHOP) FIXED;

/* CHARTYPE() IS USED TO DISTINGUISH CLASSES OF SYMBOLS IN THE SCANNER.
   TX() IS A TABLE USED FOR TRANSLATING FROM ONE CHARACTER SET TO ANOTHER.
   CONTROL() HOLDS THE VALUE OF THE COMPILER CONTROL TOGGLES SET IN $ CARDS.
   NOT_LETTER_OR_DIGIT() IS SIMILIAR TO CHARTYPE() BUT USED IN SCANNING
   IDENTIFIERS ONLY.

   ALL ARE USED BY THE SCANNER AND CONTROL() IS SET THERE.
*/
DECLARE (CHARTYPE, TX) (255) BIT(8),
        (CONTROL, NOT_LETTER_OR_DIGIT)(255) BIT(1);

/* ALPHABET CONSISTS OF THE SYMBOLS CONSIDERED ALPHABETIC IN BUILDING
   IDENTIFIERS     */
   DECLARE ALPHABET CHARACTER INITIAL ('ABCDEFGHIJKLMNOPQRSTUVWXYZ') ;

/* BUFFER HOLDS THE LATEST CARDIMAGE,
   TEXT HOLDS THE PRESENT STATE OF THE INPUT TEXT
   (NOT INCLUDING THE PORTIONS DELETED BY THE SCANNER),
   TEXT_LIMIT IS A CONVENIENT PLACE TO STORE THE POINTER TO THE END OF TEXT,
   CARD_COUNT IS INCREMENTED BY ONE FOR EVERY SOURCE CARD READ,
   ERROR_COUNT TABULATES THE ERRORS AS THEY ARE DETECTED,
   SEVERE_ERRORS TABULATES THOSE ERRORS OF FATAL SIGNIFICANCE.
*/
DECLARE (BUFFER, TEXT) CHARACTER,
   (TEXT_LIMIT, CARD_COUNT, ERROR_COUNT, SEVERE_ERRORS, PREVIOUS_ERROR) FIXED
   ;

DECLARE NUMBER_VALUE FIXED;

/* EACH OF THE FOLLOWING CONTAINS THE INDEX INTO V() OF THE CORRESPONDING
   SYMBOL.   WE ASK:    IF TOKEN = IDENT    ETC.    */
   DECLARE (IDENT, NUMBER, STRING, EOFILE, ALTEND, ENDV, SEMI,
      ASSIGN, MODV, EQLV, THENCL, FINTHCL) FIXED;

/* STOPIT() IS A TABLE OF SYMBOLS WHICH ARE ALLOWED TO TERMINATE THE ERROR
   FLUSH PROCESS.  IN GENERAL THEY ARE SYMBOLS OF SUFFICIENT SYNTACTIC
   HIERARCHY THAT WE EXPECT TO AVOID ATTEMPTING TO START CHECKING AGAIN
   RIGHT INTO ANOTHER ERROR PRODUCING SITUATION.  THE TOKEN STACK IS ALSO
   FLUSHED DOWN TO SOMETHING ACCEPTABLE TO A STOPIT() SYMBOL.
   FAILSOFT IS A BIT WHICH ALLOWS THE COMPILER ONE ATTEMPT AT A GENTLE
```

```
   RECOVERY.  THEN IT TAKES A STRONG HAND.  WHEN THERE IS REAL TROUBLE
   COMPILING IS SET TO FALSE, THEREBY TERMINATING THE COMPILATION.
*/
DECLARE STOPIT(100) BIT(1), (FAILSOFT, COMPILING) BIT(1);

DECLARE S CHARACTER;  /* A TEMPORARY USED VARIOUS PLACES */

/* THE ENTRIES IN PRMASK() ARE USED TO SELECT OUT PORTIONS OF CODED
   PRODUCTIONS AND THE STACK TOP FOR COMPARISON IN THE ANALYSIS ALGORITHM */
DECLARE PRMASK(5) FIXED INITIAL (0, 0, "FF", "FFFF", "FFFFFF", "FFFFFFFF");


/*THE PROPER SUBSTRING OF POINTER IS USED TO PLACE AN  |  UNDER THE POINT
   OF DETECTION OF AN ERROR DURING CHECKING.  IT MARKS THE LAST CHARACTER
   SCANNED.  */
DECLARE POINTER CHARACTER INITIAL ('
                                        |');
DECLARE CALLCOUNT(20) FIXED   /* COUNT THE CALLS OF IMPORTANT PROCEDURES */
   INITIAL(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);

/* RECORD THE TIMES OF IMPORTANT POINTS DURING CHECKING */
DECLARE CLOCK(5) FIXED;


/* COMMONLY USED STRINGS */
DECLARE X1 CHARACTER INITIAL(' '), X4 CHARACTER INITIAL('    ');
DECLARE PERIOD CHARACTER INITIAL ('.');

/* TEMPORARIES USED THROUGHOUT THE COMPILER */
DECLARE (I, J, K, L) FIXED;

DECLARE TRUE LITERALLY '1', FALSE LITERALLY '0', FOREVER LITERALLY 'WHILE 1';

/*  THE STACKS DECLARED BELOW ARE USED TO DRIVE THE SYNTACTIC
   ANALYSIS ALGORITHM AND STORE INFORMATION RELEVANT TO THE INTERPRETATION
   OF THE TEXT.  THE STACKS ARE ALL POINTED TO BY THE STACK POINTER SP.  */

DECLARE STACKSIZE LITERALLY '75';  /* SIZE OF STACK  */
DECLARE PARSE_STACK (STACKSIZE) BIT(8); /* TOKENS OF THE PARTIALLY PARSED
                                           TEXT */
DECLARE VAR (STACKSIZE) CHARACTER;/* EBCDIC NAME OF ITEM */
DECLARE FIXV (STACKSIZE) FIXED;   /*  ENTRY IN CONST, DICT OR IL  */
DECLARE FVTYPE (STACKSIZE) BIT(8);  /* FIXV TYPE, 0-MISCELLANEOUS,, 1-CONST,
        2-DICT,3-IL  */
DECLARE NUMDOTS (STACKSIZE) BIT(8);  /*# OF INDIRECT REFS, FROM P3 ::= .P3 */
DECLARE TEMPS (STACKSIZE) FIXED;   /* REMEMBER TEMPORARIES, ETC.  */

/* SP POINTS TO THE RIGHT END OF THE REDUCIBLE STRING IN THE PARSE STACK,
```

```
      MP POINTS TO THE LEFT END, AND
      MPP1 = MP+1.
   */
   DECLARE (SP, MP, MPP1) FIXED;



   /*                P R O C E D U R E S                                 */


PAD:
   PROCEDURE (STRING, WIDTH) CHARACTER;
      DECLARE STRING CHARACTER, (WIDTH, L) FIXED;

      L = LENGTH(STRING);
      IF L >= WIDTH THEN RETURN STRING;
      ELSE RETURN STRING || SUBSTR(X70, 0, WIDTH-L);
   END PAD;

I_FORMAT:
   PROCEDURE (NUMBER, WIDTH) CHARACTER;
      DECLARE (NUMBER, WIDTH, L) FIXED, STRING CHARACTER;

      STRING = NUMBER;
      L = LENGTH(STRING);
      IF L >= WIDTH THEN RETURN STRING;
      ELSE RETURN SUBSTR(X70, 0, WIDTH-L) || STRING;
   END I_FORMAT;

ERROR:
   PROCEDURE(MSG, SEVERITY);
      /* PRINTS AND ACCOUNTS FOR ALL ERROR MESSAGES */
      /* IF SEVERITY IS NOT SUPPLIED, 0 IS ASSUMED */
      DECLARE MSG CHARACTER, SEVERITY FIXED;
      ERROR_COUNT = ERROR_COUNT + 1;
      /* IF LISTING IS SUPPRESSED, FORCE PRINTING OF THIS LINE */
      IF ¬ CONTROL(BYTE('L')) THEN
         OUTPUT = I_FORMAT (CARD_COUNT, 4) || ' |' || BUFFER || '|';
      OUTPUT = SUBSTR(POINTER, TEXT_LIMIT-CP+MARGIN_CHOP);
      OUTPUT = '*** ERROR, ' || MSG ||
            '.  LAST PREVIOUS ERROR WAS DETECTED ON LINE ' ||
            PREVIOUS_ERROR || '.  ***';
      PREVIOUS_ERROR = CARD_COUNT;
      IF SEVERITY > 0 THEN
         IF SEVERE_ERRORS > 25 THEN
```

```
            DO;
               OUTPUT = '*** TOO MANY SEVERE ERRORS, CHECKING ABORTED ***';
               COMPILING = FALSE;
            END;
         ELSE SEVERE_ERRORS = SEVERE_ERRORS + 1;
   END ERROR;


     /*  PRINT THE PRODUCTION   */
PRODTRACE:
   PROCEDURE(P);
      DECLARE (P,K) FIXED,  L CHARACTER;
        L = '  PROD. #  '|| PRDTB(P)    ||'    '|| V(HDTB(P)) ||' ::= ';
        DO K = LEFT TO RIGHT;
         L = L || V(PARSE_STACK(K));
        END;
        OUTPUT = L;
   END PRODTRACE;



  /*                        CARD IMAGE HANDLING PROCEDURE                          */


GET_CARD:
   PROCEDURE;
      /* DOES ALL CARD READING AND LISTING                                   */
      DECLARE I FIXED, (TEMP, TEMPO, REST) CHARACTER, READING BIT(1);
            BUFFER = INPUT;
            IF LENGTH(BUFFER) = 0 THEN
               DO; /* SIGNAL FOR EOF */
                  CALL ERROR ('EOF MISSING OR COMMENT STARTING IN COLUMN 1.',1);
                  BUFFER = PAD('   EOF;   ELUDOM   EOF    ' ,80);
               END;
            ELSE CARD_COUNT = CARD_COUNT + 1;  /* USED TO PRINT ON LISTING */
      IF MARGIN_CHOP > 0 THEN
         DO; /* THE MARGIN CONTROL FROM DOLLAR | */
            I = LENGTH(BUFFER) - MARGIN_CHOP;
            REST = SUBSTR(BUFFER, I);
            BUFFER = SUBSTR(BUFFER, 0, I);
         END;
      ELSE REST = '';
      TEXT = BUFFER;
      TEXT_LIMIT = LENGTH(TEXT) - 1;
      IF CONTROL(BYTE('M')) THEN OUTPUT = BUFFER;
      ELSE IF CONTROL(BYTE('L')) THEN
         OUTPUT = I_FORMAT (CARD_COUNT, 4) || ' |' || BUFFER || '|' || REST;
```

```
      CP = 0;
   END GET_CARD;

   /*                 THE SCANNER PROCEDURES                    */


CHAR:
   PROCEDURE;
      /* USED FOR STRINGS TO AVOID CARD BOUNDARY PROBLEMS */
      CP = CP + 1;
      IF CP <= TEXT_LIMIT THEN RETURN;
      CALL GET_CARD;
   END CHAR;

  /*   ADDS MACROS TO MACRO STACK          */
  /*   SCMEDAY      */
      PUTMAC :
         PROCEDURE ;
           DECLARE S1 FIXED ;
           S1 = 0 ;
           DO WHILE S1 ¬= BYTE('$') ;  CALL CHAR ;
             S1 = BYTE(TEXT,CP) ;      /*  BYPASS MACROS  */
           END ;
           RETURN ;
         END PUTMAC ;

  /*   NUMBER CONVERSION TO CORE IMAGE, EXCEPT HEX    */
      CONSVERT :
         PROCEDURE ;
           DECLARE (S2,S3,S4) FIXED, S1 FIXED INITIAL(0) ;
           IF CTYPE(CONST) = 2 | CTYPE(CONST) = 3 THEN
           DO ;
             DO S1 = 0 TO LENGTH(CONVTEMP) - 1;
               NUMBER_VALUE = 10 * NUMBER_VALUE + BYTE(CONVTEMP,S1) - "F0" ;
             END ;
             IF CTYPE(CONST) = 2 THEN DO ;
             NUMVAL(NUMVALPTR) = NUMBER_VALUE ;
             VALPTR(CONST) = NUMVALPTR ;  NUMVALPTR = NUMVALPTR + 1 ;
             END ;  ELSE
             NUMPTR(CONST) = NUMBER_VALUE ;  RETURN ;
           END ;
           CALL ERROR(' CANNOT CONVERT REAL, LONG REAL OR PACKED YET ') ;
         RETURN ;
         END CONSVERT ;

SCAN:
   PROCEDURE;
```

```
  DECLARE (S1, S2) FIXED;
  CALLCOUNT(3) = CALLCOUNT(3) + 1;
  FAILSOFT = TRUE;
  BCD = '';  NUMBER_VALUE = 0;
SCAN1:
  DO FOREVER;
    IF CP > TEXT_LIMIT THEN CALL GET_CARD;
    ELSE
      DO; /* DISCARD LAST SCANNED VALUE */
        TEXT_LIMIT = TEXT_LIMIT - CP;
        TEXT = SUBSTR(TEXT, CP);
        CP = 0;
      END;
    /*  BRANCH ON NEXT CHARACTER IN TEXT                    */
    DO CASE CHARTYPE(BYTE(TEXT));

      /*  CASE 0  */

      /* ILLEGAL CHARACTERS FALL HERE  */
      CALL ERROR ('ILLEGAL CHARACTER: ' || SUBSTR(TEXT, 0, 1));

      /*  CASE 1  */

      /*  BLANK  */
      DO;
        CP = 1;
        DO WHILE BYTE(TEXT, CP) = BYTE(' ') & CP <= TEXT_LIMIT;
          CP = CP + 1;
        END;
        CP = CP - 1;
      END;

      /*  CASE 2  */

    DO;   /* LOCATE STRING, PLACE IN CHARST */
    CONST=CCNST+1 ;
    TOKEN=STRING ;   CTYPE (CONST) = 6 ;
    VALPTR (CONST) = CHARPTR ;   PLTIND (CONST) = 0 ;
     DC FOREVER ;
       DO CP = CP + 1 TO TEXT_LIMIT ;
       S1 = BYTE(TEXT,CP) ;   IF S1 = "7D" THEN
        DO;  CP = CP + 1 ;   NUMPTR(CONST) = CHARPTR - VALPTR(CONST) ;
             RETURN ;
        END;
       CHARST(CHARPTR) = S1 ;   CHARPTR = CHARPTR + 1 ;
       END ;
      CALL GET_CARD ;   CP = -1 ;  /* TO GET 1ST CHAR ON CARD */
     END ;
```

```
END ;
   /*  CASE 3  */
  ;  /*  PRESERVED FOR FUTURE EXPANSION  */
   /*  CASE 4  */
   DO FOREVER;  /* A LETTER:  IDENTIFIERS AND RESERVED WORDS */
      DO CP = CP + 1 TO TEXT_LIMIT;
         IF NCT_LETTER_OR_DIGIT(BYTE(TEXT, CP)) THEN
            DO;   /* END OF IDENTIFIER   */
               IF CP > 0 THEN BCD = BCD || SUBSTR(TEXT, 0, CP);
               S1 = LENGTH(BCD);
               IF S1 > 1 THEN IF S1 <= RESERVED_LIMIT THEN
                  /* CHECK FOR RESERVED WORDS *7
                  DO I = V_INDEX(S1-1) TO V_INDEX(S1) - 1;
                     IF BCD = V(I) THEN
                        DO;
                           TOKEN = I;
                           RETURN;
                        END;
                  END;
 /*  RESERVED WORDS ABOVE; THEREFORE IDENTIFIER OR MACRO */
  IF BCD = 'MACRC' THEN
    DO;  CALL PUTMAC ;   CP = CP + 1 ;
      BCD = '' ;   GO TO SCAN1 ;
    END;
/* NOT MACRO DECLARATION SO MATCH IDENT TO MACRO NAMES,
    IF IT MATCHES PROCESS MACRO, ELSE   */
               TOKEN = IDENT;
               RETURN;
            END;
      END;
      /*  END OF CARD  */
      BCD = BCD || TEXT;
      CALL GET_CARD;
      CP = -1;
   END;
   /*  CASE 5  */
DO;   /* DIGIT OR #: A NUMBER - CONVERT TO CORE IMAGE,
          ENTER IN CONSTANT TABLE */
  TOKEN = NUMBER;   CONST = CONST + 1;   PLTIND(CONST) = 0;
  VALPTR(CONST) = 0;
  IF BYTE(TEXT,CP) = BYTE('#') THEN
    DO;  /*  ITS A HEX  */
```

```
       CALL CHAR;   CTYPE(CONST) = 1;    S2 = 0;
       DO WHILE BYTE(TEXT,CP) > "C0";
        DO;    S1 = BYTE(TEXT,CP);
         IF S1 < "C7" THEN S1 = S1 - "B7"; ELSE
         IF S1 >= "F0" THEN S1 = S1 - "F0"; ELSE
         S2 = 9;  /* FORCE ERROR MESSAGE */
         NUMBER_VALUE = SHL(NUMBER_VALUE,4) + S1;
         S2 = S2 + 1;
        END;
        CALL CHAR;
       END;
       IF S2 < 9 THEN DO;  NUMVAL(NUMVALPTR) = NUMBER_VALUE;
        VALPTR(CCNST) = NUMVALPTR;   NUMVALPTR = NUMVALPTR + 1;
        END;  ELSE
       CALL ERROR ('    ILLEGAL HEX CONSTANT    ');   RETURN;
       END;   ELSE
     CONVTEMP = '' ;
   DC FOREVER;
     DO CP = CP TO TEXT_LIMIT;
       S1 = BYTE(TEXT,CP);
       IF S1 >= "F0" | S1 = "C5" | S1 = "4E" | S1 = "60" | S1 = "4B"
        THEN DO;   CONVTEMP = CONVTEMP||S1 ;
        IF S1 = "4B" THEN CTYPE(CONST) = 4; /* ITS REAL */
        END;  ELSE
       DO;    IF S1 = "C8" THEN CTYPE(CONST) = 3;  /* HALF WORD */
       ELSE IF S1 = "D7" THEN CTYPE(CONST) = 7;  /* PACKED DECIMAL */
       ELSE IF S1 = "D3" THEN CTYPE(CONST) = 5; /* LONG REAL */
       ELSE DO;  IF CTYPE(CONST) ¬= 4 THEN CTYPE(CONST) = 2; /*INTEGER */
          CP = CP - 1;
         END;
       CP = CP + 1;   CALL CONSVERT;   RETURN;
       END;
     END;
     CALL GET_CARD;
   END;
END;  /* OF NUMBERS */

      /*  CASE 6  */

DO;   /* BYPASS COMMENTS */
   S1 = BYTE(TEXT,CP);   S2 = BYTE('%');
   DO FOREVER;
     DO CP = CP + 1 TO TEXT_LIMIT;
       IF BYTE(TEXT,CP) = S2 & S1 = S2 THEN DO;
        CP = CP + 1;   GO TO SCAN1;   END;   /* A % % COMMENT  */
     END;
     CALL GET_CARD;
     IF S1 ¬= S2 THEN  GO TO SCAN1;   /* A ? COMMENT  */
```

```
         END;
      END;  /* OF COMMENTS */

            /*  CASE 7  */
            DO;       /*  SPECIAL CHARACTERS  */
               TCKEN = TX(BYTE(TEXT));
         IF BYTE(TEXT) ¬= BYTE(':')  THEN DO;  CP = 1;  RETURN;  END;
              CALL CHAR;
              IF BYTE(TEXT,CP) = BYTE('=') THEN
               DO;  CP = CP + 1;     TOKEN = ASSIGN;
               END;
               RETURN;
            END;

            /*  CASE 8  */
       ;   /* PRESEVERED FOR FUTURE EXPANSION */

         END;     /* OF CASE ON CHARTYPE  */
         CP = CP + 1;  /* ADVANCE SCANNER AND RESUME SEARCH FOR TOKEN  */
      END;
   END SCAN;


  /*                         TIME AND DATE                                */


PRINT_TIME:
   PROCEDURE (MESSAGE, T);
      DECLARE MESSAGE CHARACTER, T FIXED;
      MESSAGE = MESSAGE || T/360000 || ':' || T MOD 360000 / 6000 || ':'
         || T MOD 6000 / 100 || '.';
      T = T MOD 100;   /* DECIMAL FRACTION  */
      IF T < 10 THEN MESSAGE = MESSAGE || '0';
      OUTPUT = MESSAGE || T || '.';
   END PRINT_TIME;

PRINT_CATE_AND_TIME:
   PROCEDURE (MESSAGE, D, T);
      DECLARE MESSAGE CHARACTER, (D, T, YEAR, DAY, M) FIXED;
      CECLARE MONTH(11) CHARACTER INITIAL ('JANUARY', 'FEBRUARY', 'MARCH',
         'APRIL', 'MAY', 'JUNE', 'JULY', 'AUGUST', 'SEPTEMBER', 'OCTOBER',
         'NOVEMBER', 'DECEMBER'),
      DAYS(12) FIXED INITIAL (0, 31, 60, 91, 121, 152, 182, 213, 244, 274,
         305, 335, 366);
      YEAR = D/1000 + 1900;
      DAY = D MOD 1000;
```

```
      IF (YEAR & "3") ¬= 0 THEN IF DAY > 59 THEN DAY = DAY + 1; /* ¬ LEAP YEAR*/
      M = 1;
      DO WHILE DAY > DAYS(M);  M = M + 1;  END;
      CALL PRINT_TIME(MESSAGE || MONTH(M-1) || X1 || DAY-DAYS(M-1) ||  ', '
         || YEAR || '.  CLOCK TIME = ', T);
   END PRINT_DATE_AND_TIME;

  /*                          INITIALIZATION                                     */


INITIALIZATION:
   PROCEDURE;
      EJECT_PAGE;
   CALL PRINT_DATE_AND_TIME('  NPGS  -|-  BLISS-360  -|- -|- MOD 0          ',
      DATE_OF_GENERATION, TIME_OF_GENERATION) ;
      DOUBLE_SPACE;
      CALL PRINT_DATE_AND_TIME ('TODAY IS ', DATE, TIME);
      DOUBLE_SPACE;
     TESTIT = 1;        /*  TURN ON THE DEBUG STATEMENTS   */
      DO I = 1 TO NSY ;
         S = V(I);
         IF S = '<NUMBER>' THEN NUMBER = I;  ELSE
         IF S = '<IDENTIFIER>' THEN IDENT = I;  ELSE
         IF S = '<STR>' THEN STRING = I ; ELSE
         IF S = '_|_' THEN EOFILE = I;  ELSE
         IF S = ';' THEN DO ;  STOPIT(I) = TRUE ;  SEMI = I ;  END ;  ELSE
         IF S = '<HEADING>' THEN MODV = I;  ELSE
         IF S = '=' THEN EQLV = I;  ELSE
         IF S = ')' THEN ALTEND = I;  ELSE
         IF S = '<THEN CLAUSE>' THEN THENCL = I; ELSE
         IF S = '<FIN THEN CLAUSE>' THEN FINTHCL = I; ELSE
         IF S = 'END' THEN ENDV = I ;  ELSE
         IF S = ':=' THEN ASSIGN = I;  ELSE
         ;
      END;
      IF IDENT = NT THEN RESERVED_LIMIT = LENGTH(V(NT-1));
      ELSE RESERVED_LIMIT = LENGTH(V(NT));
      V(EOFILE) = 'EOF';
      STOPIT(EOFILE) = TRUE;
      DO I = 0 TO 255;
         NOT_LETTER_OR_DIGIT(I) = TRUE;
         CHARTYPE(I) = 0;
      END;
         CHARTYPE(BYTE(' ')) = 1;
      DO I = 0 TO LENGTH(ALPHABET) - 1;
         J = BYTE(ALPHABET, I);
         TX(J) = I;
```

```
         NOT_LETTER_OR_DIGIT(J) = FALSE;
         CHARTYPE(J) = 4;
      END;
      DO I = 0 TO 9;
         J = BYTE('0123456789', I);
         NOT_LETTER_OR_DIGIT(J) = FALSE;
         CHARTYPE(J) = 5;
         CHARTYPE(BYTE('#')) = 5 ;      /*  TO RECOGNIZE HEX  */
      END;
      DO I = V_INDEX(0) TO V_INDEX(1) - 1;
         J = BYTE(V(I));
         TX(J) = I;
         CHARTYPE(J) = 7;
      END;
      CHARTYPE(BYTE('%')) = 6 ;         /*  COMMENTS  */
      CHARTYPE(BYTE('?')) = 6 ;    /* END OF CARD COMMENTS */
      CHARTYPE("7D") = 2 ;         /*  STRING  */
      CONST = -1;    /* FOR FIRST CONSTANT ENTRY */
      I = ADDR(ADDX);     J = 1;
      DO WHILE J <= NUMBOPS;
        COREBYTE(I) = J;
        I = I + 1;    J = J + 1;
      END;
      /* FIRST SET UP GLOBAL VARIABLES CONTROLLING SCAN, THEN CALL IT */
      CP = 0;  TEXT_LIMIT = -1;
      TEXT = '';
      CONTROL(BYTE('L')) = TRUE;
      CALL SCAN;

      /* INITIALIZE THE PARSE STACK */
      SP = 1;  PARSE_STACK(SP) = EOFILE;

   END INITIALIZATION;



DUMPIT:
   PROCEDURE;     /* DUMP OUT THE STATISTICS COLLECTED DURING THIS RUN  */
      DOUBLE_SPACE;
      /*  PUT OUT THE ENTRY COUNT FOR IMPORTANT PROCEDURES */

      OUTPUT = 'STACKING DECISIONS= ' || CALLCOUNT(1);
      OUTPUT = 'SCAN               = ' || CALLCOUNT(3);
      OUTPUT = 'FREE STRING AREA   = ' || FREELIMIT - FREEBASE;
   END DUMPIT;
```

```
STACK_DUMP:
   PROCEDURE;
      DECLARE LINE CHARACTER;
      LINE = 'PARTIAL PARSE TO THIS POINT IS: ';
      DO I = 2 TO SP;
         IF LENGTH(LINE) > 105 THEN
            DO;
               OUTPUT = LINE;
               LINE = X4;
            END;
         LINE = LINE || X1 || V(PARSE_STACK(I));
      END;
      OUTPUT = LINE;
   END STACK_DUMP;


  /*                    THE SYNTHESIS ALGORITHM FOR XPL                                                    */


SYNTHESIZE:
PROCEDURE(PRODUCTION_NUMBER);
   DECLARE PRODUCTION_NUMBER FIXED;

    /*   THE INTERMEDIATE LANGUAGE TABLE  WILL BE CONSTRUCTED IN
       THIS PROCEDURE ,   THE DICTIONARY WILL ALSO BE BUILT FOR
         APPROPRIATE CONSTRUCTIONS    */


    IF PRODUCTION_NUMBER > 150 THEN GO TO SYNTH1;  /* CASE TOO BIG  */
   DO CASE PRODUCTION_NUMBER;
     ;
 /*  <MODULE> ::= <MODULE HEAD> <MODULE END>    */
   DO;
      IF MP ¬= 2 THEN  /* WE DIDN'T GET HERE LEGITIMATELY  */
         DO;
            CALL ERROR ('EOF AT INVALID POINT', 1);
            CALL STACK_DUMP;
         END;
      COMPILING = FALSE;
   END;
 /*  <MODULE HEAD> ::= <HEADING> <EQE>    */
     ;
 /*  <MODULE HEAD> ::= <HEADING> <ID CLAUSE>    */
     ;
 /*  <HEADING> ::= MODULE    */
     ;
```

```
 /*  <MODULE END> ::= ELUDOM     */
     ;
 /*  <E> ::= <LABELLED EXPR>     */
     ;

                    .                              .
                    .                              .
                    .                              .

 /*  <COROUTINE EXPRESSION> ::= <CREATE EXPRESSION>    */
     ;
   END;   /* 1ST PRODUCTION CASE  */
    RETURN;
  SYNTH1:  DO CASE PRODUCTICN_NUMBER-150;
     ;   /*  PRODUCTION # = CASE # + 150    */
 /*  <COROUTINE EXPRESSION> ::= <EXCHJ EXPRESSION>    */
     ;
 /*  <CREATE EXPRESSION> ::= <CREATE HEAD> <CREATE END>    */
     ;

                    .                              .
                    .                              .
                    .                              .

 /*  <REGISTER END> ::= <STRUCTURE ACCESS> <EQE>    */
     ;
   END;   /* 2ND PRODUCTION CASE   */
END SYNTHESIZE;


  /*               SYNTACTIC PARSING FUNCTIONS                           */


RIGHT_CCNFLICT:
   PROCEDURE (LEFT) BIT(1);
      DECLARE LEFT FIXED;
      /*  THIS PROCEDURE IS TRUE IF TOKEN IS ¬ A LEGAL RIGHT CONTEXT OF LEFT*/
      RETURN ("C0" & SHL(BYTE(C1(LEFT), SHR(TOKEN,2)), SHL(TOKEN,1)
         & "06")) = 0;
   END RIGHT_CCNFLICT;


RECCVER:
   PROCEDURE;
      IF ¬ FAILSOFT THEN CALL SCAN ;  FAILSOFT = TRUE ;
      DO WHILE FAILSOFT ;
```

```
         IF SP = 1 THEN
           DO ;  SP = SP + 1 ;  PARSE_STACK(SP) = MODV;
            SP = SP + 1;    PARSE_STACK(SP) = EQLV;
         SCANSEMI :
           DO WHILE TOKEN ¬= SEMI;    CALL SCAN;
           END ;
         CALL SCAN ;  FAILSOFT = FALSE ;
         END ;  ELSE
      IF PARSE_STACK(SP) = SEMI THEN GO TO SCANSEMI ;  ELSE
      IF PARSE_STACK(SP) = ENDV | PARSE_STACK(SP) = ALTEND THEN
         DO ;  DO WHILE (TOKEN ¬= SEMI) & (TOKEN ¬= ENDV) & (TOKEN ¬= ALTEND);
           CALL SCAN;
               END ;  FAILSOFT = FALSE ;
         END ;  ELSE SP = SP - 1 ;
      END;
      OUTPUT = 'RESUME:' || SUBSTR(POINTER, TEXT_LIMIT-CP+MARGIN_CHOP+7);
   END RECOVER;

STACKING:
   PROCEDURE BIT(1);  /* STACKING DECISION FUNCTION */
      CALLCOUNT(1) = CALLCCUNT(1) + 1;
      DC FOREVER;    /* UNTIL RETURN  */
         DO CASE SHR(BYTE(C1(PARSE_STACK(SP)),SHR(TOKEN,2)),SHL(3-TCKEN,1)&6)&3;

            /*  CASE 0  */
            DO;     /* ILLEGAL SYMBOL PAIR  */
               IF PARSE_STACK(SP) = THENCL THEN DO ; /* THEN CLAUSE FIX */
               PARSE_STACK(SP) = FINTHCL; RETURN FALSE; END;
               CALL ERROR('ILLEGAL SYMBOL PAIR: ' || V(PARSE_STACK(SP)) || X1 ||
                  V(TOKEN), 1);
               CALL STACK_DUMP;
               CALL RECOVER;
            END;

            /*  CASE 1  */

            RETURN TRUE;        /*  STACK TCKEN  */

            /*  CASE 2  */

            RETURN FALSE;       /* DON'T STACK IT YET  */

            /*  CASE 3  */

            DO;       /* MUST CHECK TRIPLES  */
               J = SHL(PARSE_STACK(SP-1), 16) + SHL(PARSE_STACK(SP), 8) + TOKEN;
               I = -1;  K = NC1TRIPLES + 1;  /* BINARY SEARCH CF TRIPLES  */
               DO WHILE I + 1 < K;
```

```
                L = SHR(I+K, 1);
                IF C1TRIPLES(L) > J THEN K = L;
                ELSE IF C1TRIPLES(L) < J THEN I = L;
                ELSE RETURN TRUE;   /* IT IS A VALID TRIPLE   */
             END;
             RETURN FALSE;
          END;

       END;     /* OF DO CASE   */
    END;    /*  OF DO FOREVER */
 END STACKING;

PR_OK:
 PROCEDURE(PRD) BIT(1);
    /* DECISION PROCEDURE FOR CONTEXT CHECK OF EQUAL OR IMBEDDED RIGHT PARTS*/
    DECLARE (H, I, J, PRD) FIXED;
    DO CASE CONTEXT_CASE(PRD);

       /*  CASE 0 -- NO CHECK REQUIRED   */

       RETURN TRUE;

       /*  CASE 1 -- RIGHT CONTEXT CHECK   */

       RETURN ¬ RIGHT_CONFLICT (HDTB(PRD));

       /*  CASE 2 -- LEFT CONTEXT CHECK   */

       DO;
          H = HDTB(PRD) - NT;
          I = PARSE_STACK(SP - PRLENGTH(PRD));
          DO J = LEFT_INDEX(H-1) TO LEFT_INDEX(H) - 1;
             IF LEFT_CONTEXT(J) = I THEN RETURN TRUE;
          END;
          RETURN FALSE;
       END;

       /*  CASE 3 -- CHECK TRIPLES   */

       DO;
          H = HDTB(PRD) - NT;
          I = SHL(PARSE_STACK(SP - PRLENGTH(PRD)), 8) + TOKEN;
          DO J = TRIPLE_INDEX(H-1) TO TRIPLE_INDEX(H) - 1;
             IF CONTEXT_TRIPLE(J) = I THEN RETURN TRUE;
          END;
          RETURN FALSE;
       END;
```

```
    END;  /* OF DO CASE  */
  END PR_OK;

 /*                          ANALYSIS ALGORITHM                                */


REDUCE:
  PROCEDURE;
    DECLARE (I, J, PRD) FIXED;
    /* PACK STACK TOP INTO ONE WORD */
    DO I = SP - 4 TO SP - 1;
      J = SHL(J, 8) + PARSE_STACK(I);
    END;

    DO PRD = PR_INDEX(PARSE_STACK(SP)-1) TO PR_INDEX(PARSE_STACK(SP)) - 1;
      IF (PRMASK(PRLENGTH(PRD)) & J) = PRTB(PRD) THEN
        IF PR_OK(PRD) THEN
        DO;  /* AN ALLOWED REDUCTION */
          MP = SP - PRLENGTH(PRD) + 1; MPP1 = MP + 1;
   IF TESTIT = 1 THEN  CALL PRODTRACE(PRD);
          CALL SYNTHESIZE(PRDTB(PRD));
          SP = MP;
          PARSE_STACK(SP) = HDTB(PRD);
          RETURN;
        END;
    END;

    /* LOOK UP HAS FAILED, ERROR CONDITION */
    CALL ERROR('NO PRODUCTION IS APPLICABLE',1);
    CALL STACK_DUMP;
    FAILSOFT = FALSE;
    CALL RECOVER;
  END REDUCE;

COMPILATION_LOOP:
  PROCEDURE;

    COMPILING = TRUE;
    DO WHILE COMPILING;      /* ONCE AROUND FOR EACH PRODUCTION (REDUCTION)  */
      DO WHILE STACKING;
        SP = SP + 1;
        IF SP = STACKSIZE THEN
          DO;
            CALL ERROR ('STACK OVERFLOW *** CHECKING ABORTED ***', 2);
            RETURN;   /* THUS ABORTING CHECKING */
          END;
```

```
            PARSE_STACK(SP) = TOKEN;
            VAR(SP) = BCD;
           FIXV(SP) = CONST;
            IF TOKEN = NUMBER THEN FVTYPE(SP) = 1; ELSE FVTYPE(SP) = 0;
            CALL SCAN;
         END;

         CALL REDUCE;
      END;     /* CF DC WHILE COMPILING  */
   END COMPILATION_LOOP;



PRINT_SUMMARY:
   PROCEDURE;
      DECLARE I FIXED;
      CALL PRINT_CATE_AND_TIME ('END OF CHECKING ', DATE, TIME);
      OUTPUT = '';
      OUTPUT = CARD_COLNT || ' CARDS WERE CHECKED.';
      IF ERROR_CCUNT = 0 THEN OUTPUT = 'NO ERRORS WERE DETECTED.';
      ELSE IF ERRCR_COUNT > 1 THEN
         OUTPUT = ERROR_COUNT || ' ERRORS (' || SEVERE_ERRCRS
            || ' SEVERE) WERE DETECTED.';
      ELSE IF SEVERE_ERRORS = 1 THEN CUTPUT = 'CNE SEVERE ERROR WAS DETECTED.';
         ELSE OUTPUT = 'ONE ERROR WAS DETECTED.';
      IF PREVICUS_ERRCR > 0 THEN
         OUTPUT = 'THE LAST DETECTED ERROR WAS ON LINE ' || PREVIOUS_ERROR
            || PERIOD;
      IF CCNTROL(EYTE('D')) THEN CALL DUMPIT;
      DOUBLE_SPACE;
      CLOCK(3) = TIME;
      DO I = 1 TO 3;    /* WATCH OUT FOR MICNIGHT */
         IF CLOCK(I) < CLOCK(I-1) THEN CLOCK(I) = CLOCK(I) +  8640000;
      END;
      CALL PRINT_TIME ('TCTAL TIME IN CHECKER     ', CLOCK(3) - CLOCK(0));
      CALL PRINT_TIME ('SET UP TIME               ', CLOCK(1) - CLOCK(0));
      CALL PRINT_TIME ('ACTUAL CHECKING TIME      ', CLOCK(2) - CLOCK(1));
      CALL PRINT_TIME ('CLEAN-UP TIME AT END      ', CLOCK(3) - CLOCK(2));
      IF CLOCK(2) > CLOCK(1) THEN   /* WATCH OUT FOR CLOCK BEING OFF */
      OUTPUT = 'CHECKING RATE: ' || 6000*CARD_COUNT/(CLOCK(2)-CLOCK(1))
         || ' CARDS PER MINUTE.';
   END PRINT_SUMMARY;

MAIN_PROCEDURE:
   PROCEDURE;
      CLOCK(0) = TIME;  /* KEEP TRACK CF TIME IN EXECUTION */
      CALL INITIALIZATION;
```

```
      CLOCK(1) = TIME;

      CALL COMPILATI CN_LOOP;

      CLOCK(2) = TIME;

      /* CLOCK(3) GETS SET IN PRINT_SUMMARY */
      CALL PRINT_SUMMARY;

   END MAIN_PROCEDURE;

CALL MAIN_PROCEDURE;
RETURN SEVERE_ERRORS;

EOF EOF EOF
```

BIBLIOGRAPHY

1. Zavoyski, E. M., Preliminary Design of the Systems Implementation Language BLISS-360, M.S. Thesis, Naval Postgraduate School, Monterey, 1972.

2. Wulf, W. A., and others, BLISS Reference Manual, Pittsburgh: Carnegie-Mellon University, Department of Computer Science, 1970.

3. McKeeman, W. M., Horning, J. J., and Wortman, D. B., A Compiler Generator, Prentice-Hall, 1970

4. Malcolm, M. A., PL360 (Revised) A Programming Language for the IBM-360, Stanford: Stanford University, Computer Science Department, May, 1971.

5. International Business Machines Corporation, IBM System/360 Operating System Assembler Language, 6th ed.

6. PDP-10 Reference Handbook, Digital Equipment Corporation, Maynard, Mass.

7. International Business Machines Corporation, IBM System/360 Principles of Operation, 8th ed.

8. Kildall, G. A., Miscellaneous Programs for CS4113, Naval Postgraduate School, Monterey, September, 1971.

9. Gries, D., Compiler Construction for Digital Computers, Wiley, 1971.

10. Wulf, W. A., Russell, D. B., and Habermann, A. N., "BLISS: A Language for Systems Programming," Communications of the ACM, v. 14, no. 12, p. 780-790, December, 1971.

11. Wile, D. S., and Geschke, C. M., "Efficient Data Accessing in the Programming Language BLISS," in Proceedings of a Symposium on Data Structures in Programming Languages, p. 306-320, (SIGPLAN Notices, v. 6, no. 2, February, 1971).

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Asst Professor G. A. Kildall, Code 53Kd Department of Mathematics Naval Postgraduate School Monterey, California 93940	1
4.	Major Richard Charles Bahler, USMCR 137 South River Street Plains, Pennsylvania 18705	1

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*
Naval Postgraduate School
Monterey, California 93940

2a. REPORT SECURITY CLASSIFICATION
Unclassified

2b. GROUP

3. REPORT TITLE
Steps Toward a Compiler for BLISS-360

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*
Master's Thesis; June 1972

5. AUTHOR(S) *(First name, middle initial, last name)*
Richard Charles Bahler

6. REPORT DATE
June 1972

7a. TOTAL NO. OF PAGES
100

7b. NO. OF REFS
11

8a. CONTRACT OR GRANT NO.

b. PROJECT NO.

c.

d.

9a. ORIGINATOR'S REPORT NUMBER(S)

9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*

10. DISTRIBUTION STATEMENT
Approved for public release; distribution unlimited.

11. SUPPLEMENTARY NOTES

12. SPONSORING MILITARY ACTIVITY
Naval Postgraduate School
Monterey, California 93940

13. ABSTRACT

The design of a compiler for the IBM S/360 systems implementation language BLISS-360, a modification of the PDP-10 language BLISS-10, is described. The compiler has a two-pass structure that is based upon the XPL Compiler Generator System. The first of these passes, which uses the XPL prototype compiler Skeleton, is examined in some detail. Fundamental data structures are described for this pass, including a constant table, a dictionary for variable definitions, and an intermediate language table to retain the source program structure and semantics. Modifications which allow the Skeleton compiler to perform a syntax analysis of BLISS-360 programs are discussed and demonstrated.

General requirements are defined for the functions to be performed by the second pass, including machine language code generation from the intermediate language, storage allocation and building program interface linkage.

DD FORM 1 NOV 65 1473 (PAGE 1)
S/N 0101-807-6811

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
XPL Compiler Generator System						
Programming Language						
BLISS						
Compiler						

Thesis 135034
B1367 Bahler
c.1 Steps toward a compiler for BLISS-360.